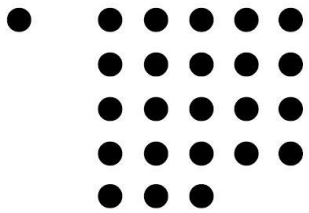

Fachhochschule Köln
Cologne University of Applied Sciences

Ein agiles Vorgehensmodell anhand von APEX 4.2

Bachelorarbeit

Vorgelegt an der Fachhochschule Köln
Campus Gummersbach
Im Studiengang Wirtschaftsinformatik

Ausgearbeitet von: Artur Kusmierczyk

Erster Prüfer: Prof. Dr. Heide Faeskorn-Woyke
Zweiter Prüfer: Prof. Dr. Birgit Bertelsmeier

Gummersbach, im August 2013

Inhaltsverzeichnis

Vorwort 1

1. Klassische Vorgehensmodelle 2

1.1 Wasserfallmodell 3

1.2 V-Modell 5

1.3 Rational Unified Process 6

2. Das Agile Manifest 8

2.1 Agile Vorgehensmodelle 9

2.1.1 Scrum 9

2.1.2 Feature Driven Development 10

2.1.3 Extreme Programming 13

2.1.4 Crystal Methods 15

2.1.5 Kanban 16

3. Scrum 17

3.1 Scrum Rollen 18

3.1.1 Product Owner 18

3.1.2 Scrum Master 18

3.1.3 Team 19

3.2 Product Backlog 20

3.3 Sprints 24

3.3.1 Sprint Planungssitzung 25

3.3.2 Sprint Backlog 28

3.3.3 Daily Scrum 29

3.3.4 Sprint Review 30

3.3.5 Sprint-Retrospektive 30

3.3.6 Sprint-Burndown-Bericht 31

3.4 Projekterfolg 33

4. Oracle Application Express (APEX) 35

4.1 Geschichte und Entwicklung bis 4.2 36

4.2 Architektur und Technik 37

4.2.1 Kommunikation Web Browser und APEX 38

4.2.2 APEX-Engine 39

4.2.3 Workspaces 40

4.3 Das APEX Framework 41

4.4 Aufbau einer APEX Anwendung 43

4.5 JavaScript, jQuery, Ajax, Dynamic Actions und die APEX API 45
5. APEX Alternativen 49
6. Fazit 50
7. Abkürzungsverzeichnis 51
8. Literaturverzeichnis 53

Abbildungsverzeichnis

Abbildung 1: Wasserfallmodell aus [Ludewig] Seite 147 3
Abbildung 2: V-Modell aus [Hindel] Seite 16. 5
Abbildung 3: Rational Unified Process 6
Abbildung 4: Featur Driven Development [WikiFDD] 10
Abbildung 5: Color Modeling 11
Abbildung 6: Crystal Methods 15
Abbildung 7: Kanban Board 16
Abbildung 8: Scrum Überblick aus [Pichler] Seite 7 17
Abbildung 9: Kano Bewertung [Glogler] Seite 137 22
Abbildung 10: Relatives Gewicht [Glogler] Seite 139 23
Abbildung 11: Planungsglas [Pichler] 26
Abbildung 12: Sprint Backlog [Pichler] 28
Abbildung 13: Sprint-Burndown-Bericht[Pichler] 31
Abbildung 14: Chaos Report aus [WikiChaos] 33
Abbildung 15: APEX Historie [Czarski] 36
Abbildung 16: APEX Architektur [Oracle2] 37
Abbildung 17: APEX Framework 41

Vorwort

Diese Arbeit betrachtet die Entwicklung einer Oracle Application Express Anwendung (APEX) unter Zuhilfenahme eines agilen Vorgehensmodells in Form von Scrum. Ziel dieser Arbeit ist die Bewertung agiler Methoden im APEX Umfeld.

Hintergrund dieser Arbeit ist ein Projekt, das an der Fachhochschule Köln für den Fachbereich Datenbanken in APEX umgesetzt wurde. Das Projektziel war die Erstellung eines administrativen Tools zur Pflege eines Multiple Choice Test für die Professoren.

Das erste Kapitel geht auf klassische Vorgehensmodelle wie das Wasserfallmodell und das V-Modell von der Projektplanung bis hin zu ihrer Fertigstellung ein.

Das zweite Kapitel befasst sich mit agilen Vorgehensmodellen. Hier wird auf unterschiedliche agile Prozesse kurz eingegangen, so dass man einen Überblick über die aktuellen Methoden erhält.

Das dritte Kapitel beschreibt detailliert den Scrum Prozess als agiles Vorgehensmodell. Es werden die beteiligten Rollen mit ihren Aufgaben sowie die im Entstehungsprozess anfallenden Artefakte beschrieben.

Das vierte Kapitel geht auf das Framework Oracle APEX ein. Als Einstieg dient die geschichtliche Entwicklung von APEX. Danach wird die technische Seite betrachtet. Im Anschluss wird der Aufbau einer APEX Anwendung erklärt. Die Betrachtung von Webtechnologien schließen dieses Kapitel ab.

Das fünfte Kapitel betrachte Alternativen zu APEX und zeigt deren Vor- und Nachteile auf.

Das sechse Kapitel enthält eine Schlussbetrachtung für die Eignung zur Einführung des Scrum Prozesses innerhalb APEX.

1. Klassische Vorgehensmodelle

Bevor auf die einzelnen klassischen Vorgehensmodelle eingegangen wird, soll zunächst der Begriff Vorgehensmodell erläutert werden.

Ein Vorgehensmodell ist, bedingt dadurch, dass es ein Modell ist, eine vereinfachte Beschreibung eines Softwareprozesses. Prozessmodelle umfassen Tätigkeiten, Softwareprodukte und die dabei beteiligten Rollen der Personen.[1]

Alle Vorgehensmodelle unterteilen sich in vier grundlegende Phasen:

- Analyse
- Entwurf beziehungsweise Implementierung
- Test
- Integration beziehungsweise Betrieb

Unterschiede ergeben sich in der Anzahl der Beteiligten Rollen, der Art und Anzahl der entstehenden Softwareprodukte und dem Umfang der dabei entstehenden Dokumente.

Allen klassischen Vorgehensmodellen ist gemein, dass sie auf umfangreiche Dokumente (Pflichtenheft, Lastenheft, Anforderungsdokumentation, Dokumentation des Grobentwurfs, Dokumentation des Feinentwurfs usw.) aufbauen.

Vorgehensmodelle zeichnen sich durch ihre Struktur und Transparenz für die an einem Softwareprojekt beteiligten Personen aus und unterstützen so den erfolgreichen Projektabschluss.

[1] Vgl.[Sommerville] Seite 24

1.1 Wasserfallmodell

Das Wasserfallmodell unterteilt sich in mehrere Phasen wie System-Analyse, Software-Spezifikation, Architekturentwurf, Feinentwurf und Codierung, Integration und Test, Installation und Abnahme und zuletzt dem Betrieb und der Wartung.

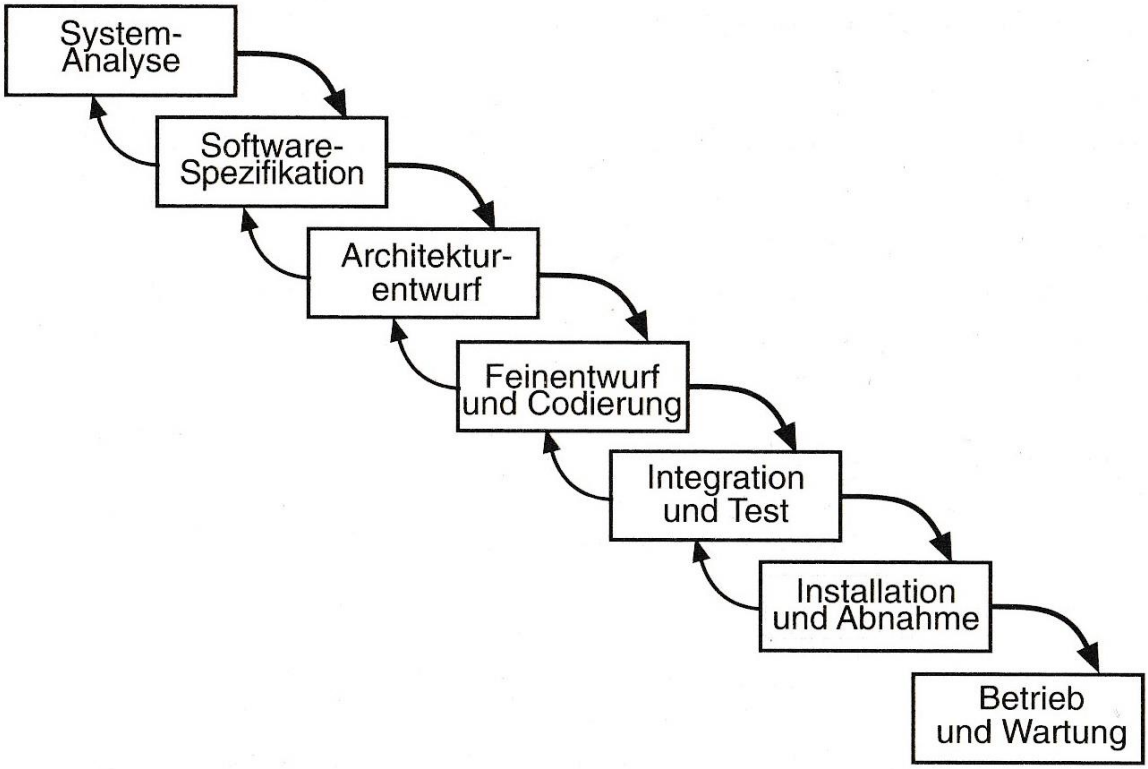

Abbildung 1: Wasserfallmodell aus [Ludewig] Seite 147

Beginnend von links oben bis nach rechts unten werden diese einzelnen Phasen durchlaufen. Da ein Softwareentwicklungsprozess lebendig und ständiger Änderung unterworfen ist, der sich aus veränderten Kundenwünschen ergibt, gibt es zusätzlich die Möglichkeit, schrittweise in vorherige Phasen zu springen. Das Ergebnis des Wasserfallmodells sind immer Dokumente am Ende einer Phase.[1]

Da das Wasserfallmodell ein starres und schwerfälliges Vorgehensmodell ist, wird sehr viel Zeit mit der Systemanalyse und der Softwarespezifikation verbracht. Diese Phasen können mehrere Monate in Anspruch nehmen, so dass wenig Zeit zur Implementierung des Programmcodes übrig bleibt.

[1] Vgl. [Ludewig] Seite 147

Das Wasserfallmodell findet heute noch in meist größeren Softwareprojekten Verwendung. Dabei definiert sich die Größe eines Softwareprojekts durch die Mannstärke, die Dauer, das Budget oder den Funktionsumfang der Software.

Nach [Hindel] ergeben sich folgende Vor- und Nachteile des Wasserfallmodells:

Vorteile:

- Einfach verständlich
- Kontrollierbarer Prozessablauf durch Meilensteine und Dokumente
- Organisatorisch gut beherrschbar
- Wenig Managementaufwand

Nachteile:

- Dokumente können einen höheren Stellenwert bekommen
- Risiken werden spät in der Implementierung oder in Tests bekannt
- Übliche Veränderungen und Detaillierungen von Anforderungen bleiben unberücksichtigt
- Das System ist erst nach vollständiger Fertigstellung einsatzbereit
- Tests sind erst dann möglich, wenn die Entwicklung abgeschlossen ist
- Keine iterative Vorgehensweise, da Entwicklungsschritte in voller Breite durgeführt werden

1.2 V-Modell

Die Erweiterung des Wasserfallmodells ist das V-Modell, das Wert auf Qualitätssicherung legt.[1] Es wurde um die Begriffe Verifikation und Validierung erweitert, die in der rechten Seite zu finden sind.

Die Verifikation hat das Ziel der Überprüfung des Softwareprodukts hinsichtlich seiner Anforderung.

Die Validierung hat das Ziel der Überprüfung des Softwareprodukts hinsichtlich seiner Funktionalität.

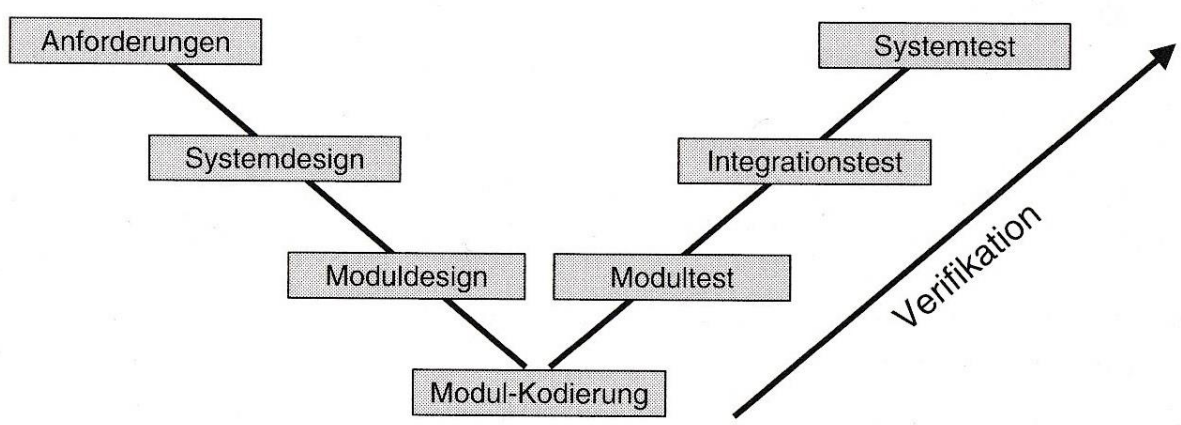

Abbildung 2: V-Modell aus [Hindel] Seite 16

Die linke Seite des V-Modells entspricht im Wesentlichen den Stufen des Wasserfallmodells wie System-Analyse, Software-Spezifikation, Architekturentwurf, Feinentwurf und Codierung. Die Testfälle werden in der linken Seite des V-Modells erstellt und der rechten Seite gegenübergestellt. So können Tests bereits sehr früh in der Anforderungsermittlung erstellt und die Softwarequalität dadurch und im Laufe des Softwareentwicklungsprozesses kontinuierlich erhöht werden.

Beim Wasserfallmodell werden die Testfälle erst am Ende des Softwareentwicklungsprozesses erstellt. So entstandene Fehler werden erst sehr spät erkannt, was zu steigenden Kosten führt. Diesem Umstand versucht das V-Modell durch die gleichzeitigen Testfälle entgegen zu wirken.

[1] Vgl. [Hindel] Seite 16

1.3 Rational Unified Process

Der Rational Unified Process (RUP) ist ein Prozessmodell der Firma Rational (jetzt IBM) für die objektorientierte Entwicklung mit der UML (Unified Modeling Language).[1]

Der RUP unterteilt sich in Phasen, die sich wiederum in mehrere Iterationen aufteilen. Die neun Workflows (Business Modeling, Requirements, Analysis and Design, Implementation, Test, Deployment, Configuration Management, Project Management und Environment) haben dabei abhängig von der jeweiligen Phase eine unterschiedliche Gewichtung.

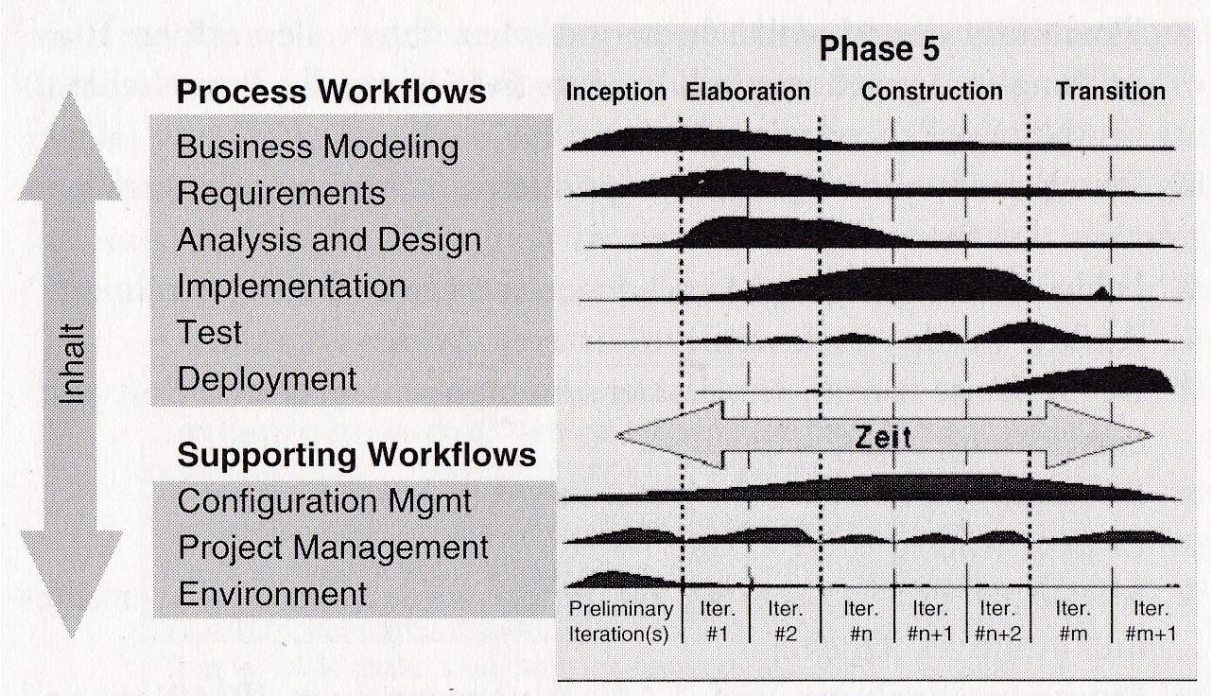

Abbildung 3: Rational Unified Process

Die dabei entstehenden Artefakte sind Dokumente, Use-Case-Diagramme, Modelle und Modellelemente.

Der RUP unterteilt sich in vier Phasen: Inception, Elaboration, Construction und Transition.[2]

Die Inception-Phase dient der konzeptuellen Anforderungsermittlung, die die wesentlichen Funktionen beschreibt.

[1] Vgl. [Hindel] Seite 19
[2] Vgl. [WikiRup]

Die Elaboration-Phase dient der Erstellung des Architekturprototyps und einer detaillierten Beschreibung für etwa 80 Prozent der Anwendungsfälle.

Die Construction-Phase dient zur Entwicklung und des Testen des Softwareprodukts.

Die Transition-Phase dient der Auslieferung des Softwareprodukts.

Nach [Hindel] ergeben sich folgende Vor- und Nachteile:

Vorteile:

- Flexibilität des Modells
- Iteratives Vorgehen optimiert den Prozess
- Frühe Erkennung von Fehlern und ungeeigneten Alternativen
- Risiken werden früh erkannt
- Eine ausführliche Sammlung von Best Practices (Anforderungsmanagement, iterative Entwicklung, Architekturorientierung, visuelle Modellierung, Qualitätskontrolle, Change- und Konfigurationsmanagement)
- Unterstützung von parallelen Aktivitäten
- Mit seiner Präsentation als Hypertext fördert RUP verteiltes und intuitives Arbeiten.
- Prozessbeschreibung und CASE-Werkzeuge sind miteinander integriert
- Artefakte sind durch CASE-Werkzeuge immer aktuell

Nachteile:

- Hoher Managementaufwand
- Sehr Komplex
- Höherer Aufwand für Iterationsplanung
- Werkzeugunterstützung herstellerabhängig (IBM/Rational)

2. Das Agile Manifest

2001 trafen sich 17 Entwickler in Utha und formulierten das Agile Manifest[1], das gemeinsame Werte festlegte und leichte Prozesse beschrieb. Dabei einigte man sich auf vier wesentliche Werte[2], die so zu verstehen sind: obwohl die Werte auf der rechten Seite ihren Wert haben, werden die Werte auf der linken Seite höher bewertet. Diese vier Werte lauten wie folgt:

Individuen und Interaktionen stehen vor Prozessen und Werkzeugen.

Prozesse und Werkzeuge unterstützen den Projekterfolg, sind jedoch nicht maßgeblich für seinen Erfolg. Entscheidend sind die beteiligten Personen mit ihrer Erfahrung und die Kommunikation innerhalb des Teams.

Funktionierende Software steht über umfangreicher Dokumentation.

Eine umfangreiche Dokumentation ist sehr zeitaufwendig und verlängert den Entwicklungsprozess. Deshalb sollte nur so viel wie nötig des Programmcodes dokumentiert werden. Denn letztendendes erwartet der Kunde eine lauffähige und einsatzbereite Software. Je früher diese fertiggestellt ist, desto positiver wirkt es sich auf den ROI[3] aus.

Die Zusammenarbeit mit dem Kunden über der Verhandlung von Verträgen.

Der Kunde soll bei der Softwareentwicklung stärker einbezogen werden. So soll frühzeitig auf sich ändernde Softwareanforderungen reagiert werden können. Es ist verständlich, dass sich beide Parteien - die Entwickler auf der einen, der Kunde auf der anderen Seite - sich absichern wollen, falls etwas schief gehen sollte. Aber das gemeinsame Ziel sollte ein Softwareprodukt sein, das beide Seiten zufrieden stellt. Aus diesem Grunde ist eine enge Zusammenarbeit nur vorteilhaft.

Das Reagieren auf Veränderungen steht über dem Befolgen eines Plans.

Das Befolgen eines festgelegten Plans lässt sich in der Realität nicht immer einhalten, da es hin und wieder zu unvorhersehbaren Ereignissen kommen kann wie zum Beispiel der krankheitsbedingte Ausfall eines Mitarbeiters. Auch können sich während der Entwicklung geänderte Anforderungen ergeben, die das Erreichen des Plans gefährden. Ein flexibles Reagieren auf Veränderungen ist demnach vorzuziehen.

[1] Vgl. [Manifesto]
[2] Vgl. [Glogler] Seite 18
[3] Return on Investment ist eine Kennzahl, die die Rentabilität ausdrückt.

2.1 Agile Vorgehensmodelle

2.1.1 Scrum

Scrum wurde erstmals in den 90er Jahren von Ken Schwaber und Jeff Sutherland eingesetzt. Es kennt nur drei wesentliche Rollen und die Anforderungen sind in einem Product Backlog hinterlegt. Scrum ist ein iteratives Vorgehen mit Zyklen von maximal 30 Tagen, den sogenannten Sprints. Dieser Prozess ist Schwerpunkt dieser Bachelorarbeit und wird im dritten Kapitel detailliert vorgestellt.

2.1.2 Feature Driven Development

Feature Driven Development (FDD) wurde von Jeff De Luca 1997 als eine agile Methode definiert.[1] Features sollen einen Mehrwert für den Kunden generieren und stellen diese in den Mittelpunkt. Basis ist ein Feature-Plan. Der Chefarchitekt behält dabei den Überblick über die Gesamtarchitektur und die fachlichen Kernmodelle.

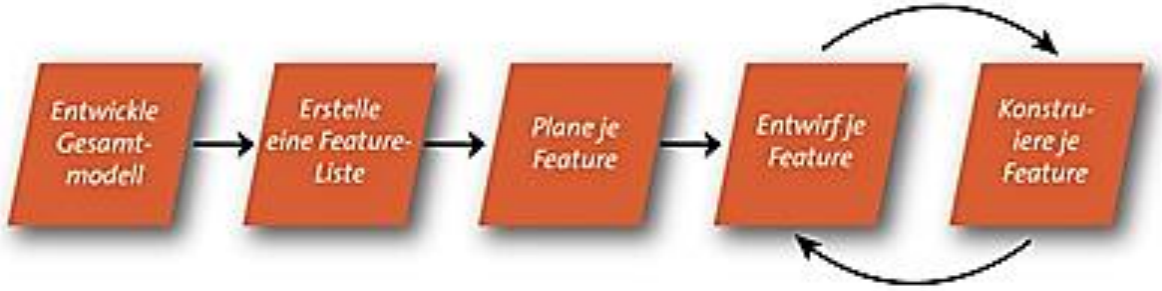

Abbildung 4: Featur Driven Development [WikiFDD]

FDD-Projekte unterteilen sich in fünf Prozesse:[2]

1. Entwickle ein Gesamtmodell

In der ersten Phase werden die einzelnen Fachmodelle des Systems von Fachexperten und Entwicklern unter der Leitung des Chefarchitekten erstellt. Ziel dieser Phase ist ein Überblick über das zu entwickelnde System zu erhalten. Dafür wird das Color Modeling verwendet.

Das Color Modeling basiert auf dem UML-Klassenmodell, das durch vier Farben die fachlichen Klassen visualisiert.

Dabei unterscheiden sich die Farben wie folgt:

- *Moment-Interval* (pink): Hier drunter Fallen ein Zeitpunkt oder Zeitraum hinsichtlich eines Geschäftsvorfalls. Zum Beispiel: Auftrag, Vorgang, Versicherungsantrag oder Produktionsprozess.
- *Role* (gelb): Hier drunter Fallen Personen oder Gegenstände. Zum Beispiel: Lieferant, Kunde, Produkt oder Artikel.
- *Description* (blau): Hier drunter Fallen katalogähnliche Einträge, die ein Objekt klassifizieren. Zum Beispiel: Produktgruppe, Artikeltyp oder Kundensegment.

[1] Vgl. [WikiFDD]
[2] Vgl. [IT-Agile4]

- *PartyPlaceThing* (grün): Hier drunter Fallen Parteien, Orte oder Dinge. Zum Beispiel: juristische Person (Partei), Unternehmen (Partei), Filiale (Ort), Ware (Ding) oder Versicherungsbedingungen (Ding).

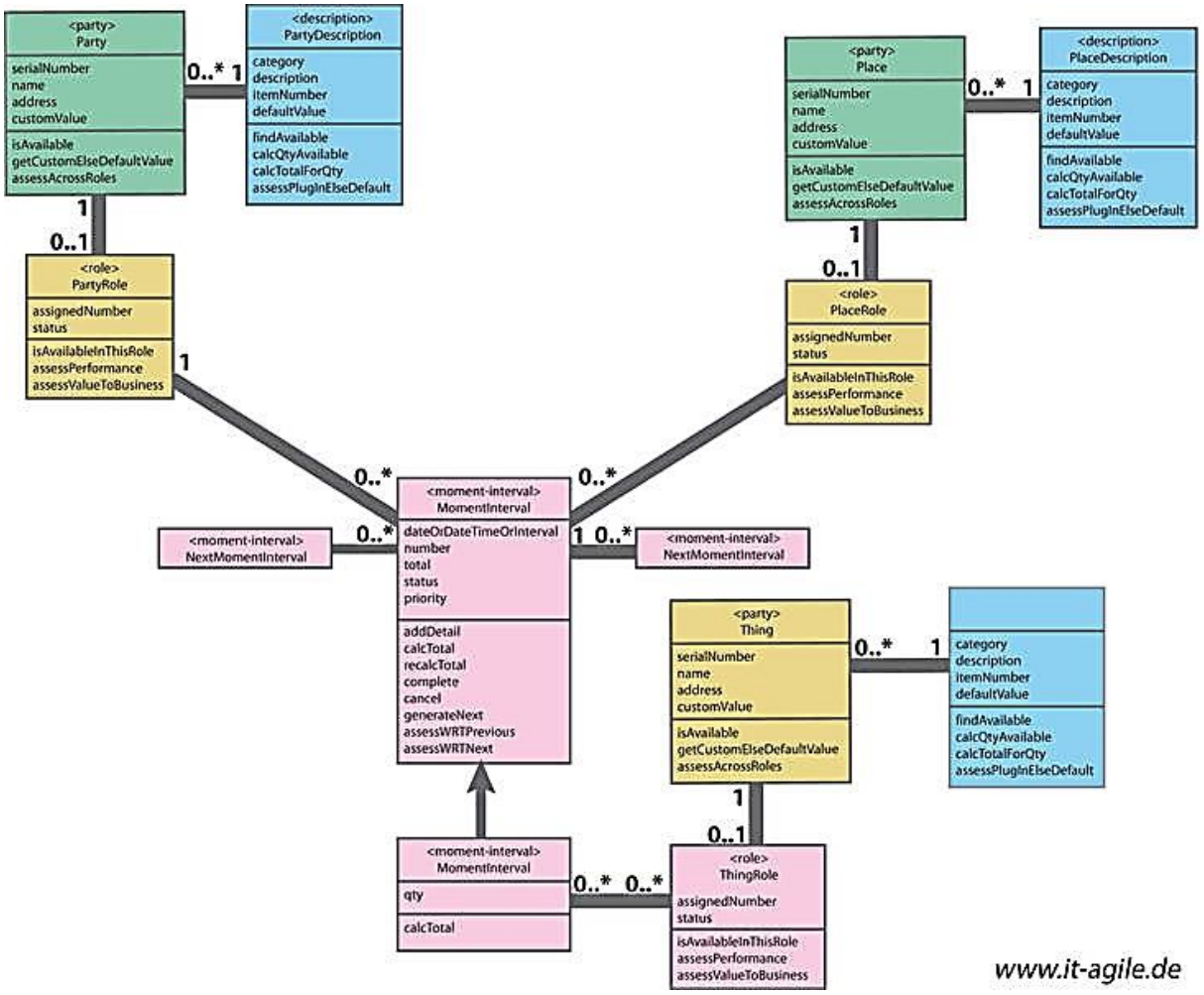

Abbildung 5: Color Modeling

2. Erstelle eine Feature-Liste

In der zweiten Phase werden die vorher festgelegten Fachgebiete in Features in einem dreistufigen Schema detailliert. Features werden nach dem Schema *Aktion, Ergebnis, Objekt* beschrieben, zum Beispiel *Berechne Gesamtsumme der Verkäufe.* Das Ergebnis ist eine kategorisierte Feature-Liste.

3. Plane je Feature

In der dritten Phase wird die Reihenfolge der Features festgelegt. Die Reihenfolge richtet sich nach den Abhängigkeiten zwischen den Features, der Auslastung des Teams und der Komplexität der Features. Darauf basierend werden die Fertigstellungstermine je Geschäftsaktivität festgelegt.

4. Entwirf je Feature

In der vierten Phase werden die Sequenzdiagramme der einzelnen Features erstellt und so die Klassenmodelle verfeinert. Darauf basierend werden die Klassen und Methodenrümpfe geschrieben und die daraus resultierenden Ergebnisse zur Qualitätssicherung wiederum inspiziert.

5. Konstruiere je Feature

In der fünften Phase werden die Features programmiert. Dabei werden Codeinspektion und Komponententests zur Qualitätssicherung eingesetzt.

2.1.3 Extreme Programming[1]

Extreme Programming (XP) wurde von Ken Beck, Ward Cunningham und Ron Jeffries bei der Erstellung von Software bei Chrysler entwickelt und erfolgreich angewandt. Dabei stehen Teamarbeit, Offenheit und Kommunikation im Vordergrund.

Oft sind am Anfang eines Softwareprojekts die Anforderungen des Kunden und die Informationen für die Entwickler nicht hinlänglich bekannt und Prioritäten können sich sehr schnell ändern. Diese Risiken stehen im Vordergrund von XP.

Um diese Risiken zu minimieren, wird jede Iteration, also das Entwickeln einer neuen Funktionalität, einer Risikoanalyse, Nutzenanalyse, Prototyping und Akzeptanz unterworfen, bevor die Entwicklung, Integration und das Testen stattfindet.

Durch die Einbeziehung des Kunden kann dieser frühzeitig gegensteuern, falls das erwartete Ergebnis von seiner Vorstellung abweicht.

Beim XP werden unterschiedliche Praktiken angewendet:

- Pair-Programming. Beim Pair-Programming wird paarweise programmiert. Dabei sollte der Wissensstand zwischen den Entwicklern nicht zu sehr voneinander abweichen, da die Kommunikationskosten steigen würden. Der Vorteil beim Pair-Programming ist der Wissenstransfer zwischen den Entwicklern.
- Testgetriebene Entwicklung. Bei der Testgetriebenen Entwicklung werden im Vorfeld sogenannte Unit-Test erstellt. Erst dann wird mit dem Erstellen des Codes begonnen. Dadurch soll die Softwarequalität steigen.
- Kollektives Eigentum. Dadurch, dass Entwickler beim Pair-Programming nach einer gewissen Zeit rotieren, soll vermieden werden, dass Entwickler Teile ihres Programmcodes als ihr „Eigentum“ ansehen.
- Permanente Integration. Durch kurze Iterationen erhält man ein lauffähiges Softwaresystem.
- Kundenbeziehungen. Durch die enge Zusammenarbeit mit dem Kunden lassen sich frühzeitig die gewünschten Anforderungen herauskristallisieren. Dies geschieht in der Regel mit User-Storys.
- Refactoring. Durch ständige Reviews wird der Programmcode optimiert und vorhandene Fehler identifiziert.
- Keine Überstunden. Überstunden werden vermieden, da sich eine Überlastung der Entwickler negativ auf die Produktivität auswirkt.

[1] Vgl. [WikiXP]

- Iterationen. Durch kurze Iterationen erhalten die Entwickler ein schnelles Feedback über die umgesetzten Anforderungen.
- Metapher. Durch ein Glossar werden sprachliche Missverständnisse vermieden. Entwickler und Kunden sprechen oft eine andere Sprache, d.h. das Vokabular des Kunden ist dem Entwickler nicht geläufig und umgekehrt.
- Coding-Standarts. Das Entwicklerteam einigt sich im Vorfeld auf festgelegte Programmierstandarts. So ist der Programmcode für jeden verständlich nachvollziehbar.
- Einfaches Design. Überflüssiger Programmcode soll vermieden werden. Oft werden Funktionen hinzugefügt, die keinen Mehrwert generieren und so einen erheblichen Zeit- und Kostenaufwand darstellen.
- Planning-Game. Entwickler und Kunden versuchen gemeinsam in einem Planing-Poker die Anforderungen zu identifizieren. Kunden wissen, was für sie wichtig ist und können so ihre Anforderungen priorisieren. Entwickler können den dafür benötigten Aufwand sehr gut einschätzen. Das Ergebnis ist eine priorisierte Liste der Anforderungen.

2.1.4 Crystal Methods[1]

Bei einer Befragung von erfolgreichen Softwareprojekten erstellte Alistair Cockburn in den 90er Jahren eine Matrix, die sich nach der Größe der beteiligten Entwickler und den Risiken richtet. Auf dieser Grundlage sollten entsprechende Vorgehensmodelle beziehungsweise Prozesse gewählt werden, die sich in Crystal Clear, Crystal Yellow, Crystal Orange, Crystal Orange Web, Crystal Red, Crystal Magenta und Crystal Blue unterteilen. Die Farbe spiegelt dabei die Anzahl der Personen wider.

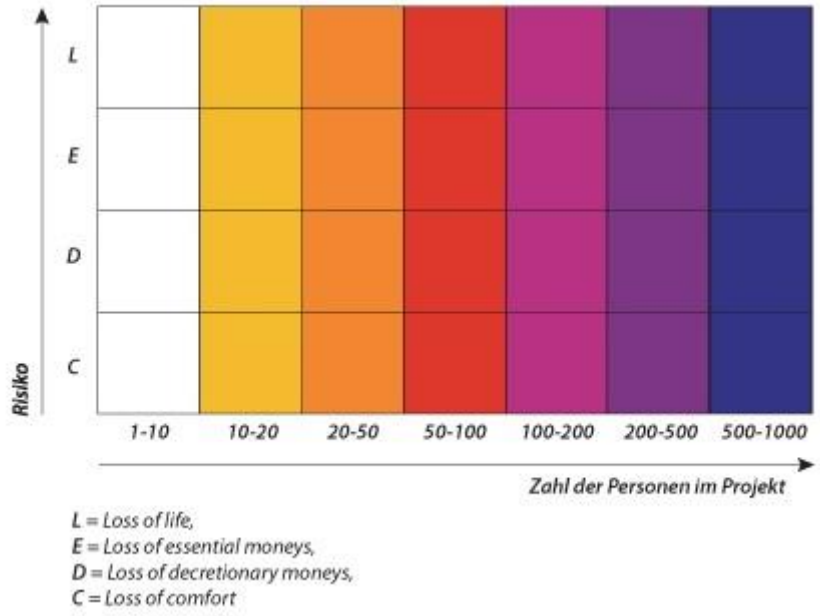

Abbildung 6: Crystal Methods

[1] Vgl. [IT-Agile1]

2.1.5 Kanban[1]

Kanban ermöglicht die Visualisierung des Projektfortschritts durch ein sogenanntes Kanban Board. Dies ist in der Regel ein White-Board, auf dem die einzelnen Funktionalitäten im Backlog[2] durch Zettel festgehalten werden. Die einzelnen Projektschritte wie Bereit, Entwickeln, Testen und Release haben eine Limitierung des WIP[3]. Dies soll das Team vor Überlastung schützen. Sobald ein Projektschritt ausgelastet ist, werden keine neuen Funktionalitäten aufgenommen. Der Workflow läuft nach dem Pull Prinzip ab, das heißt, dass neue Funktionalitäten erst dann aufgenommen werden, wenn es der WIP zulässt. Dabei durchlaufen die Zettel die einzelnen Schritte von links nach rechts.

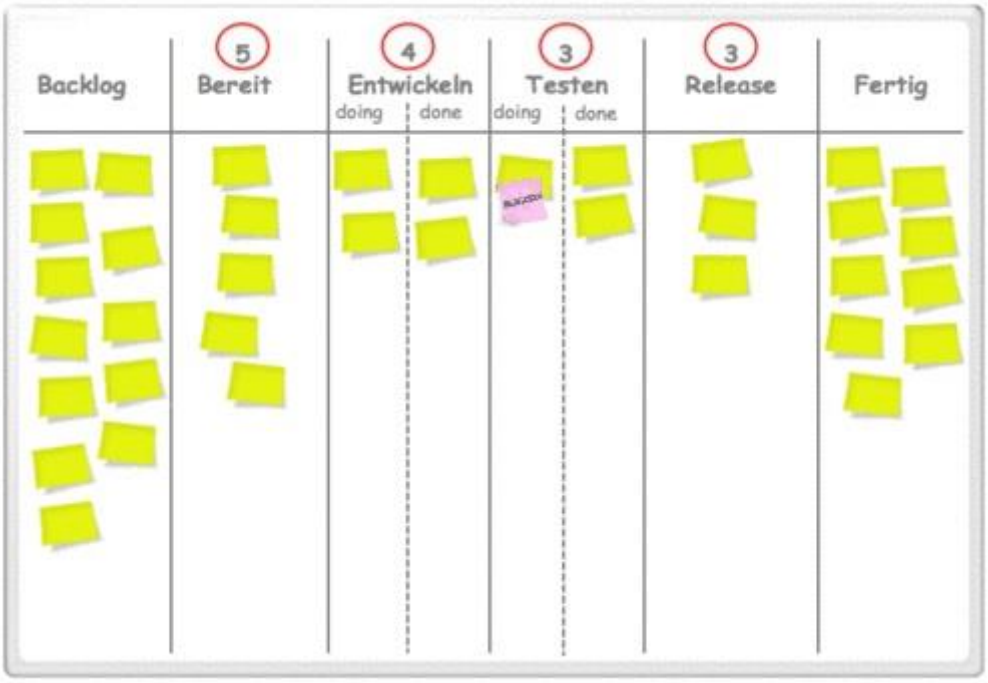

Abbildung 7: Kanban Board

[1] Vgl. [IT-Agile2]
[2] Das Backlog enthält die umzusetzenden Anforderungen.
[3] Der Work in Progress drückt die Menge der parallelen Aufgaben aus.

3. Scrum

Ein Scrum-Projekt besteht aus kurzen Zyklen, den sogenannten Sprints, die maximal 30 Tage dauern. Häufig werden kürzere Zyklen von bis zu einer Woche gewählt.[1] Ausgangspunkt ist der Product Backlog, der priorisierte Anforderungen enthält. Zu Beginn des Sprints erstellt das Team in einer Sprint-Planungssitzung den Sprint Backlog, in dem die zu implementierenden Anforderungen ausgewählt werden. In einem Daily Scrum bespricht das Team eventuelle Hindernisse und koordiniert die anstehenden Aufgaben. Der Daily Scrum sollte immer am selben Ort und zu einer festgelegten Uhrzeit stattfinden. Nun kann die eigentliche Entwicklungsarbeit beginnen. Am Ende eines Sprints findet ein Sprint-Review statt, das die tatsächlich umgesetzten Anforderungen mit denen aus der Sprint-Planungssitzung vergleicht. Im Anschluss an das Sprint-Review findet die Sprint-Retrospektive statt, die die Zusammenarbeit innerhalb des Teams fördern soll. Das Ergebnis soll ein auslieferbares Produktinkrement zum Ziel haben.[2]

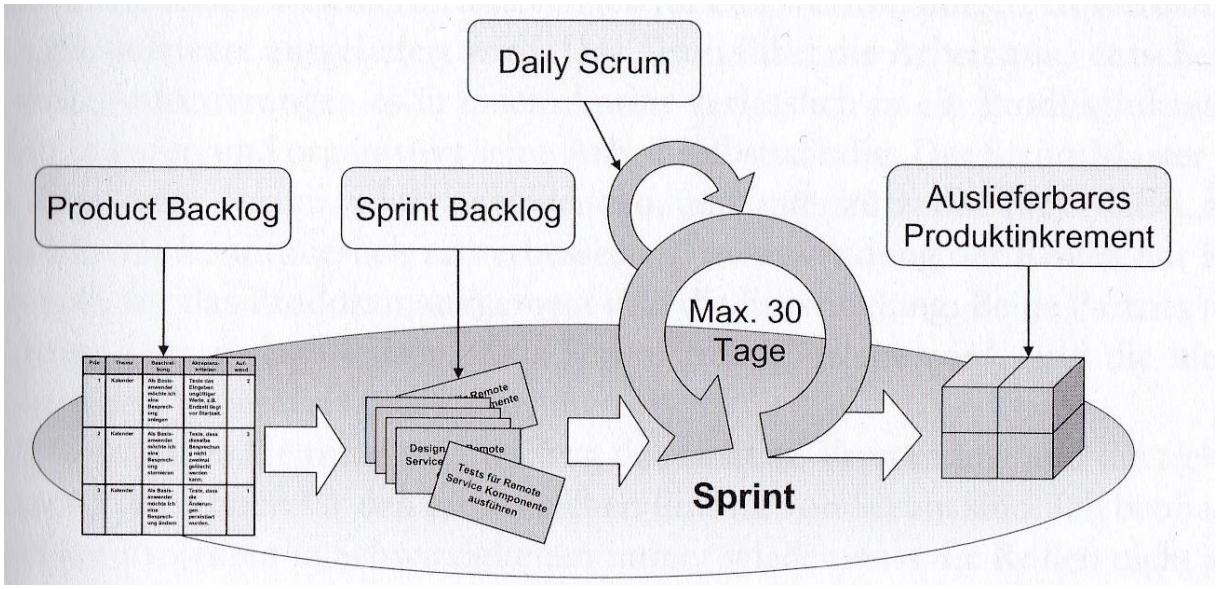

Abbildung 8: Scrum Überblick aus [*Pichler*] Seite 7

[1] Vgl. [Pichler] Seite 62
[2] Vgl. [Pichler] Seite 7

3.1 Scrum Rollen

Scrum kennt nur drei wesentliche Rollen: den Product Owner, den Scrum Master und das Team. Der Product Owner ist für die Anforderungen und die Auslieferung der Software zuständig. Der Scrum Master sorgt für die richtige Anwendung des Scrum Prozesse. Das Team entscheidet, wie viele Anforderungen in einem Sprint umsetzbar sind und organisiert seine Arbeit selbstständig.[1]

3.1.1 Product Owner

Der Product Owner ermittelt die Kundenanforderungen und gibt diese an das Team weiter. Seine Rolle lässt sich somit als Schnittstelle zwischen Auftraggeber und Entwicklerteam beschreiben. Dabei erstellt er gemeinsam mit dem Kunden die gewünschten Anforderungen und hält diese priorisiert nach ihrer Dringlichkeit im Product Backlog fest. Er ist maßgeblich für den wirtschaftlichen Erfolg verantwortlich. Seine Rolle wird häufig von einem Produktmanager oder Marketingmitarbeiter gefüllt.[2]

3.1.2 Scrum Master

Die Rolle des Scrum Masters lässt sich als Coach und Change Agent beschreiben.[3] Seine Aufgabe ist, dass das Team und der Product Owner ihre Aufgaben im Scrum Prozess richtig verstehen und leben. Er hilft dem Product Owner beim Priorisieren des Product Backlog und schützt das Team vor äußeren Einflüssen, die den Entwicklungsprozess behindern können wie zum Beispiel das Abziehen von Entwicklern oder die Überlastung der Entwickler durch Spezialaufgaben. Der Scrum Master wird vom Team gewählt und ist idealerweise mit der Entwicklung von Softwareprojekten vertraut.

Zudem versucht er die Produktivität des Teams in jedem Sprint zu steigern und die Qualität des Softwareprodukts zu erhöhen.[4]

Der Scrum Master ist für die Lösung aller Hindernisse, sogenannte Impediments, die das Team bei ihrer Arbeit behindern, verantwortlich.

[1] Vgl. [Pichler] Seite 9
[2] Vgl. [Pichler] Seite 10
[3] Vgl. [Pichler] Seite 19
[4] Vgl. [Glogler] Seite 95

3.1.3 Team

[Glogler] beschreibt zwei typische Arten von Teams bei (meist großen) Unternehmen.[1] Dem Standart-Software-Team und das Software-Projektteam.

Das Standart-Software-Team besteht überwiegend nur aus Software-Entwicklern, die gemeinsam an einer Applikation arbeiten oder zeitgleich bei mehreren unterschiedlichen Softwareprojekten mitwirken. In beiden Fällen wissen die Teammitglieder nicht, woran der andere arbeitet. Sie verbringen viel Zeit in Meetings, so dass jeder maximal zwei bis drei Stunden am Tag entwickelt. Feste Arbeitszeiten lassen sich nicht erkennen. Die einen arbeiten sehr früh, andere arbeiten nachts. Für bestimmte Teile der Applikation gibt es Spezialisten. Fehlen diese, muss die Fachabteilung auf neue Funktionalitäten warten. Sobald die Funktionalitäten erfolgreich umgesetzt wurden, wird an die Tester übergeben.

Das Software-Projektteam arbeitet an einem Projekt, das meist über mehrere Jahre geht. Glogler bemängelt hier unter anderem die monatelange Erstellung der Fachspezifikation, unvollständige und ungenaue Anforderungen, das Fehlen von entscheidenden Projektverantwortlichen, Code wird neu geschrieben, da ein neuer Entwickler den bereits vorhanden nicht versteht und Tests gehen schief. Am Ende ist nur eine fehlerhafte Software entstanden.

Hier unterscheidet sich die Softwareentwicklung in Scrum von den oben beschriebenen sehr deutlich.

Die Softwareentwicklung in Scrum ist stark teambasiert.[2] Alle wichtigen Rollen arbeiten eng zusammen wie zum Beispiel Datenbankentwickler, Architekten, Tester und Softwareentwickler. Das Team entscheidet selbst, wie viele Anforderungen im nächsten Sprint aufgenommen werden und organisiert seine Arbeit selbstständig. Auch kann es zu neuen Anforderungen nein sagen, wenn die Auslastung zu hoch ist. Denn es verpflichtet sich bei jedem Sprint die dabei ausgewählten Anforderungen in ein lauffähiges Produktinkrement umzusetzen. Üblicherweise sind Scrum Teams zwischen fünf und neun Mann groß. Zu große Teams würden die Kommunikationskosten steigen lassen und das Fortschreiten des Entwicklungsprozesses würde ins Stocken geraten. Das Team sollte in einem gemeinsamen Arbeitsraum arbeiten, da so ein Kern agiler Softwareentwicklung - Kommunikation und Kollaboration – effektiv eingesetzt werden kann.

[1] Vgl. [Glogler] Seite 69
[2] Vgl. [Pichler] Seite 13

3.2 Product Backlog

Das Product Backlog enthält alle Anforderungen und Arbeitsergebnisse, die zur Zielerreichung des Projekts notwendig sind.[1] Es wird vom Product Owner erstellt, der im Vorfeld mit dem Kunden die gewünschten Anforderungen ermittelt hat. Da alle Anforderungen bei Projektstart nicht immer klar identifiziert werden können, wird zunächst ein grober Product Backlog erstellt, der mit jedem Sprint verfeinert wird. So werden neue Anforderungen hinzugefügt, detailliert oder gestrichen. Die Anforderungen werden nach Nutzen, Risiko und Kosten vom Product Owner zu Beginn eines jeden Sprints priorisiert. Je wichtiger eine Anforderung ist, desto detaillierter ist seine Anforderungsbeschreibung.

Jedoch sei hier zu erwähnen, dass das Product Backlog keine Anforderungen im klassischen Sinne enthält, sondern nur Eigenschaften und Merkmale beschreibt.[2] [Glogler] nennt sie deshalb Backlog Items.

Diese Anforderungen werden in *user stories* [3]festgehalten und beschreiben die Anforderungen aus der Anwendersicht.[4] Die user story besteht aus einem Namen, der Beschreibung der Anforderung und Akzeptanzkriterien. Akzeptanzkriterien sind Kriterien, die für die Anforderung erfüllt sein müssen, damit sie erfolgreich realisiert wurde.

Ein Beispiel zur Verdeutlichung wie eine user story aussehen könnte: „Als Buchhalter würde ich gerne alle unbezahlten Rechnungen sehen, damit ich Mahnungen erstellen kann."

Zur Priorisierung des Product Backlog stellt [Glogler] vier Methoden vor, die hier kurz erklärt werden.[5]

Die erste Methode ist die *MusCoW* Methode. MusCoW ist eine Zusammensetzung aus den Worten *Must*, *Could* und *Wish* und spiegelt die Dringlichkeit der Funktionalität wider. Zu Must gehören alle Funktionalitäten, die elementar und vom Entwicklerteam unbedingt umzusetzen sind. Could enthält alle Nice-To-Have Features. Es wäre schön, wenn man sie hat, sie sind aber nicht von hoher Dringlichkeit. Wish enthält die Funktionalitäten, die man gerne hätte, die aber nur noch auf der Wunschliste des Product Owners stehen und nicht mehr ausschlaggebend sind.

[1] Vgl. [Pichler] Seite 27
[2] Vgl [Glogler] Seite 124
[3] Benutzergeschichten
[4] Vgl. [Pichler] Seite 48
[5] Vgl. [Glogler] Seite 134

Die zweite Methode nennt Glogler die 1000-Ping-Pong-Bälle. Man kann sie auch die 500 Äpfel nennen. Man wählt eine Einheit aus und setzt eine Basismenge an. Diese Basismenge wird nun auf die Backlog Items verteilt. Je mehr Ping-Pong-Bälle ein Item erhält, desto wertvoller kann es sein. Im nächsten Schritt kann man mit dem Kano Modell herausarbeiten, ob man mit der Einschätzung richtig lag.

Das dritte Modell ist das Kano Modell. Es hilft bei der Klassifizierung von Kundenwünschen und teilt diese in sechs Kategorien ein:

1. Must haves. Darunter fallen Basisfunktionalitäten.
2. Excitors. Dies sind Merkmale, die man nicht erwartet, aber toll findet.
3. Lineare Merkmale. Je mehr davon vorhanden ist, desto besser.
4. Indifferent. Für den Kunden unwichtige Merkmale.
5. Questionable. Alle Merkmale, bei denen nicht klar ist, ob sie gebraucht werden.
6. Reverse Attribute. Merkmale, die so vom Kunden nicht gewollt sind.

Um herauszufinden, welches Merkmal in welche Kategorie fällt, wird ein Fragebogen erstellt und eine kleine Gruppe von 20 bis 30 Personen befragt. Jedes Merkmal enthält zwei Fälle. Im funktionalen Fall ist das Merkmal vorhanden, im dysfunktionalen Fall ist das Merkmal nicht vorhanden. In beiden Fällen kann zwischen folgenden Antworten gewählt werden:

1. Ich mag das Merkmal in dieser Art.
2. Ich erwarte das Merkmal genau so.
3. Ich bin neutral gegenüber diesem Merkmal.
4. Ich kann mit dem Merkmal leben.
5. Ich mag das Merkmal in dieser Form nicht.

Nun wird eine Matrix aus den funktionalen und dysfunktionalen Fragen erzeugt und den jeweiligen Kategorien zugeordnet. Zur besseren Verständlichkeit ein Beispiel:

Wurde in der funktionalen Frage die Antwort „Ich erwarte das Merkmal genau so.“ und in der dysfunktionalen Frage die Antwort „Ich mag das Merkmal in dieser Form nicht.“ gewählt, so wird das Merkmal der Kategorie „Must haves“ zugeordnet.

So erhält man für jedes Merkmal pro Befragten eine Antworte, in welche der Kategorien es fällt. Abschließend wird eine Tabelle erstellt, die alle Antworten aller Befragten aggregiert. Die resultierende Kategorie ergibt sich aus den maximal aggregierten Merkmalen. Zur Veranschaulichung dient folgende Tabelle:

Merkmal	Gesamt	E	L	M	I	R	Q	Resultierende Kategorie
Live View	30	10	8	3	5	2	2	Excitor
Blitz auf Gehäuse	30	5	5	7	5	3	3	Must-have
Sensorreinigung	30	6	4	10	4	0	6	Must-have

Abbildung 9: Kano Bewertung [Glogler] Seite 137

Der Vorteil der Kano-Methode ist, dass der Nutzer befragt wird. Dies setzt natürlich die Zusammenarbeit und Kommunikation von Product Owner, Team und Anwender voraus.

Die vierte Methode ist die Relative-Weight-Methode nach Karl Wiegers, die sich in 8 Schritte teilt:

Im ersten Schritt werden alle Backlog Items in einer Liste festgehalten.

Im zweiten Schritt schätzt man den relativen Vorteil eines jeden Backlog Items in einer Skala von 1 bis 9. 1 steht für den geringsten, 9 für den höchsten Vorteil aus Businness-Sicht.

Im dritten Schritt wird die relative Strafe ebenfalls in einer Skala wie bei Schritt zwei geschätzt. Sie gibt an, wie hoch die Strafe ist, wenn eine Funktionalität fehlt.

Im vierten Schritt werden relativer Vorteil und relative Strafe aufaddiert. Nun wird der Gesamtbetrag prozentual am Business Value ermittelt.

Im fünften Schritt werden die relativen Kosten anhand von geschätzten Storypoints erfasst. Storypoints ist eine Maßeinheit, die die Größe eines Backlog Items ausdrückt. Damit man Backlog Items nach ihrem Aufwand eindeutiger unterscheiden kann, hat Mike Cohn in Anlehnung an die Fibonacci-Reihe die nach ihm benannte „Cohnsche Unreine-Fibonacci-Reihe“ erstellt, die von der agilen Community zur Ermittlung von Aufwandshöhe bei Schätzungen der Backlog Items herangezogen wird.[1] Die Aufwandshöhe kann hier die Werte 0, 1, 2, 3, 5, 8, 13, 20, 40, 100 usw. annehmen und ist aussagekräftiger als zum Beispiel die Einteilung der Aufwandshöhe in Werte von 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, usw. Sobald die relativen Kosten ermittelt wurden, wird ihr prozentualer Anteil an den Gesamtkosten ermittelt.

Im sechsten Schritt werden die relativen Risiken in einer Skala von 1 bis 9 bewertet und ihr prozentualer Anteil an den Gesamtrisiken ermittelt.

[1] Vgl. [Glogler] Seite 143

Im siebten Schritt wird die Prioritätszahl P mit der folgenden Formel berechnet

$$P = \frac{\text{Gesamtbetrag in Prozent}}{\text{Kosten in Prozent} + \text{Risiko in Prozent}}$$

Im achten Schritt wird nun das Product Backlog nach der Prioritätszahl P absteigend sortiert.

Das priorisierte Product Backlog könnte so ausschauen:

Merkmal	Relativer Vorteil	Relative Strafe	Gesamt-betrag BV	BV in Prozent	Relative Kosten [SP]	Kosten in Prozent	Relatives Risiko	Risiko in Prozent	Prioritäts-zahl
Live View	5	3	8	32	8	15	1	6	1,48
Sensorreinigung	3	1	4	16	13	25	3	19	0,37
Blitz auf Gehäuse	6	2	8	32	20	38	4	25	0,50
Großer Monitor	2	1	3	12	8	15	6	38	0,23
Gesichtserkennung	1	1	2	8	3	6	2	13	0,44
Total		**8**	**25**	**100**	**52**	**100**	**16**	**100**	**5**

Abbildung 10: Relatives Gewicht [Glogler] Seite 139

3.3 Sprints

Sprints sind kurze Zyklen, die Anforderungen iterativ in ausführbare Produktinkremente wandeln.[1] Maximal dauern Sprints 30 Tage, sie können aber auch im Wochenrhythmus stattfinden. Vor jedem Sprint findet eine Sprint-Planungssitzung statt, in der das Team die Anforderungen und deren Anzahl auswählt. Täglich findet zur selben Zeit und am gleichen Ort ein Daily Scrum statt. Am Ende eines Sprints finden ein Sprint-Review und eine Sprint-Retrospektive statt.

Zudem ist allen Sprint Meetings gemein, dass sie timeboxed sind. Das heißt, sie finden immer im festgelegten zeitlichen Rahmen statt. Jedes Meeting und jeder Sprint haben eine klare Deadline.[2]

Innerhalb des Sprints findet die eigentliche Entwicklung statt. Eine Forderung agiler Prozesse ist funktionierende und fehlerfreie Software. Dazu bietet sich die Methode der testgetriebenen Entwicklung (engl. Test-Driven Developement, kurz TDD) an.[3]

Zuerst werden Testfälle geschrieben, von denen der Entwickler glaubt, sie könnten relevant für die zu implementierende Funktionalität sein. Diese Testfälle schlagen zunächst alle fehl. Im nächsten Schritt wird der Code geschrieben, so dass alle Testfälle fehlerfrei bestehen. Im letzten Schritt wird der Code aufgeräumt, so dass nichts Unnötiges im Code enthalten ist, das später negative Auswirkung auf die Codequalität hat.

Glogler merkt an, dass neben den Unit Tests auch noch Regressions-, Integrations-, Systemintegrations- und Akzeptanztests durchgeführt werden müssen.[4] Da dies sehr Aufwendig ist, ist die Länge eines Sprints vom Vermögen des Teams abhängig. In den ersten Sprints ist die Velocity[5] des Teams langsam. Doch mit zunehmender Erfahrung und besserer automatisierter Test-Prozesse steigt die Effektivität und Produktivität des Teams in den folgenden Sprints.

In Java gibt es beispielsweise das bekannte JUnit Framework, das Klassen und Methoden mit Unit Tests überprüft.[6]

Für relationale Datenbanken bietet sich DbUnit[7] an, das auf JUnit aufbaut.

---

[1] Vgl. [Pichler] Seite 81
[2] Vgl. [Glogler] Seite 157
[3] Vgl. [IT-Agile3]
[4] Vgl. [Glogler] Seite 231
[5] Entwicklungsgeschwindigkeit
[6] http://junit.org
[7] http://dbunit.sourceforge.net/

3.3.1 Sprint Planungssitzung

In der Sprint Planungssitzung wird vom Team ein realistischer Sprint Backlog erstellt, in dem die umzusetzenden Anforderungen ausgewählt werden und am Ende des Sprints ein Produktinkrement entsteht.[1]

Product Owner, Team und Scrum Master nehmen immer an der Sprint Planungssitzung teil. Endanwender und Manager können ebenfalls teilnehmen, haben jedoch kein Mitspracherecht und sind nur Beobachter.

Bereits im Vorfeld bereitet der Product Owner Anforderungen vor und stellt sie dem Team vor. Das Team wählt aus, welche und wie viele der Anforderungen innerhalb des Sprints umsetzbar sind. Falls Fragen oder nachträgliche Änderungen vorhanden sind, kann sich das Team an den Product Owner wenden und so gemeinsam eine Lösung finden. Der Scrum Master nimmt hierbei eine moderierende Funktion ein.

Nach [Pichler] unterscheidet man zwischen einem klassischen und iterativen Vorgehen.

Beim klassischen Vorgehen bespricht das Team gemeinsam mit dem Product Owner das Sprint-Ziel mit den vorausgewählten Anforderungen. Das Team selektiert im Anschluss die von ihm umsetzbaren Produktinkremente. Im Zweiten Schritt identifiziert das Team die dafür notwendigen Aktivitäten. Zu den Aktivitäten gehören Design-, Implementierungs-, Test- und Dokumentationsaufgaben.

Beim iterativen Vorgehen wird zunächst ebenfalls gemeinsam das Sprint-Ziel definiert. Beginnend mit der höchstprioren werden die vorbereiteten Anforderungen durch das Team iteriert. Dabei werden für jede Anforderung Aktivitäten identifiziert und abgeschätzt. Wenn das Wissen und die Team Kapazität für die Umsetzung aller Anforderungen ausreicht, werden die Anforderungen Teil der Sprint-Verpflichtung und die Aktivitäten Bestandteil des Sprint-Plans.

[Pichler] empfiehlt das iterative Vorgehen, da so das Team keine Zeit und Energie in Diskussionen von Anforderungen investiert, die auf Grund der vorhandenen Kapazitäten oder des Leistungsvermögens des Teams nicht ausführbar sind.

Zur Ermittlung eignen sich Karteikarten als strukturiertes Brainstorming-Verfahren. Sobald die Aktivitäten identifiziert sind, schätzt das Team den Aufwand in Nettopersonenstunden ab.

Aktivitäten mit höchstprioren Anforderungen werden zu Erst umgesetzt.

[1] Vgl. [Pichler] Seite 93

Wenn alle Aktivitäten identifiziert sind, überprüft das Team, ob genügend Verfügbarkeit und Wissen für die Umsetzung vorhanden ist. Zur Überprüfung der Teamkapazität ist die Technik des Planungsglases hilfreich.[1] Im Laufe der Planungssitzung werden die Aktivitäten in das Planungsglas gekippt. Ist es voll, so können keine neuen Anforderungen hinzugefügt werden. Um das Team nicht vollständig auszulasten, sollte man das Planungsglas nicht vollständig füllen, da bei Überlastungen kein Spielraum für notwendige Verbesserungsarbeiten wie Refaktorisierung vorhanden ist.

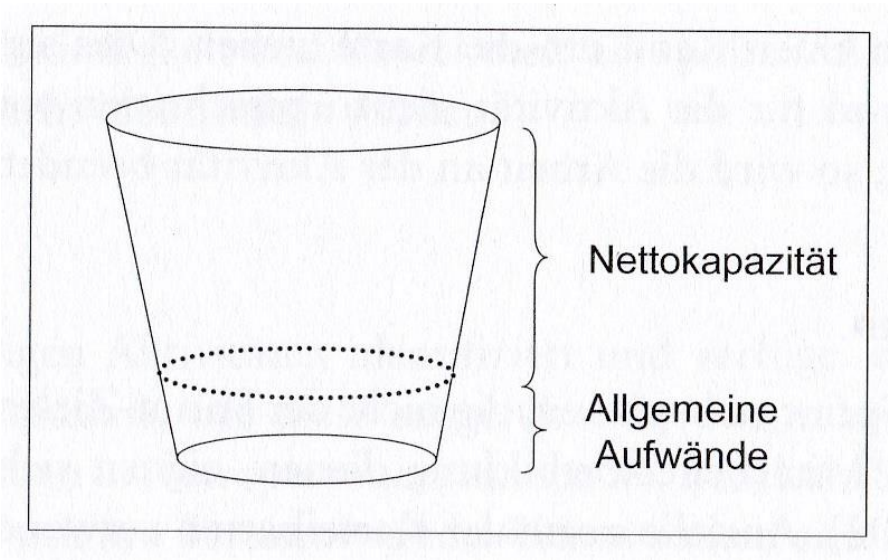

Abbildung 11: Planungsglas [Pichler]

Am Ende der Planungssitzung sollten die Ergebnisse noch einmal kurz zusammengefasst werden.

[Pichler] empfiehlt folgende Punkte bei der Planungssitzung:

Da das Team selbstorganisierend ist, sollte der Scrum Master vermeiden für das Team zu planen.

Hin und wieder kann es vorkommen, dass einzelne Teammitglieder dominieren. Teammitglieder, die wenig zur Diskussion beitragen, sollten nicht vernachlässigt werden. Da Scrum äußerst kommunikativ ist, sollten diese Personen zur Teilnahme animiert werden.

Zu viele Diskussionen über das Design des Softwaresystems sollte vermieden werden, da dies für eine effektive Planungssitzung hinderlich ist. Zudem ist es ein Hinweis, dass das Team die Anforderungen nicht verstanden hat. Er empfiehlt daher das Ausführen von Explorationsaktivitäten. Unter Explorationsaktivitäten versteht man das Erstellen von

[1] Vgl. [Pichler] Seite 98

Prototypen oder das generieren von Wissen. Sie finden vor dem Standartsprint im Explorationssprint statt.[1]

Der Product Owner darf keine Aktivitäten identifizieren oder in die Bevollmächtigungen des Teams eingreifen. Dies ist nur die Aufgabe des Teams.

Falls der Product Owner an der Planungssitzung nicht teilnehmen kann, so sollte diese verschoben werden, wenn kein Stellvertreter ernannt wurde. Durch die fehlende Diskussion mit dem Product Owner könnte das Team Anforderungen mit ihren Akzeptanzkiterien fehlinterpretieren. So würde Software entstehen, die nicht den Anforderungen entspräche und Mehrkosten verursachen würde. Stattdessen sollte das Team bis zur nächsten Planungssitzung Verbesserungsarbeiten wie Refaktorisierung oder die Verbesserung von Entwicklungs- und Testumgebung durchführen.

Der Scrum Master sollte das Team auf vage oder zu grobe Aktivitäten hinweisen, da die damit verbundenen Schätzungen zu ungenau werden. Die Aktivitäten sollten nicht größer als ein Nettotag sein.

[1] Vgl. [Pichler] Seite 55

3.3.2 Sprint Backlog

Das Sprint Backlog ist ein kollektives Zeitmanagementsystem, das alle umzusetzenden Aktivitäten enthält und es unterstützt das Team bei der Organisation der Sprint-Zielerreichung.[1]

Für bessere Transparenz und Zugänglichkeit empfiehlt sich das Verwenden einer Stellwand mit Karteikarten. Da Scrum Teams üblicherweise in einer gemeinsamen Arbeitsumgebung arbeiten, sieht jedes Teammitglied den aktuellen Projektfortschritt.

Jedes Teammitglied hält auf einer Karteikarte seine Aktivität fest, schätzt den dafür benötigten Arbeitsaufwand in Stunden ab und notiert ihn. Am Ende des Tages schätzen alle Teammitglieder den verbleibenden Restaufwand ab aktualisieren ihre Karteikarten.

Es empfiehlt sich die Unterteilung der Stellwand in Spalten wie Prio, Anforderung, Zu erledigen, In Arbeit und Erledigt. Die Spalte Prio steht für die Priorität der Anforderung. Die Spalte Anforderung hält die umzusetzende Anforderung fest. Die Spalte Zu erledigen hält alle dafür benötigten Aktivitäten fest. Die Spalte In Arbeit enthält die Karteikarte eines Teammitgliedes, der gerade die Aktivität bearbeitet. Die Spalte Erledigt hält alle umgesetzten Aktivitäten fest. So wandern während des Projekts die Aktivitäten von links nach rechts und die Anforderungen von oben nach unten.

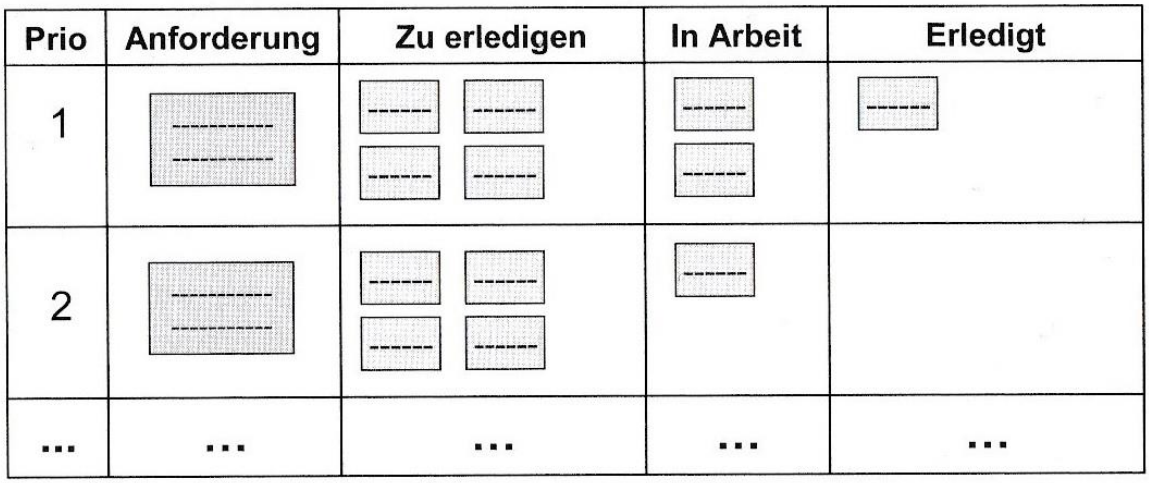

Abbildung 12: Sprint Backlog [Pichler]

[1] Vgl. [Pichler] Seite 102

3.3.3 Daily Scrum

Die Daily Scrum ist eine 15 Minütige Besprechung, die am gleichen Ort zur gleichen Zeit stattfindet und dient zur Findung von Hindernissen[1]. An ihr nehmen Product Owner, Scrum Master und das ganze Team teil.

Dabei sollen drei Fragen von jedem Teammitglied beantwortet werden:

1. Welche Aktivitäten habe ich seit der letzten Daily Scrum abgeschlossen?
2. Woran plane ich bis zur nächsten Daily Scrum zu arbeiten?
3. Werde ich in irgendeiner Form an der Ausführung einer Aktivität behindert?

Die Hindernisse werden vom Scrum Master auf einem Flip Chart mit Datum festgehalten.

[Pichler] nennt fünf Techniken für den Ablauf der Daily Scrum:

Im Stand-up Meeting bilden die Teilnehmer einen Kreis

Mit einem *speech token*, einem Gegenstand wie Stift oder Ball darf nur derjenige Reden, der den Gegenstand in der Hand hält. So soll ein reibungsloser und konfliktfreier Kommunikationsablauf gewährleistet werden.

Falls eine Diskussion unruhig verläuft, so sollte der Scrum Master diese zunächst unterbrechen und eine Metadiskussion starten. Dabei sollen die Beteiligten Vorschläge für einen ruhigen und reibungslosen Diskussionsablauf machen. Einigt man sich auf Verbesserungsmaßnahmen, so wird der Daily Scrum fortgesetzt.

Ist immer noch keine vernünftige Diskussion möglich, so sollte man den Daily Scrum zunächst verschieben.

Damit der Daily Scrum nicht zu monoton wird, sollte man später eine vierte Frage hinzufügen wie zum Beispiel „Was mache ich am Wochenende?“.

[1] Vgl. [Pichler] Seite 104

3.3.4 Sprint Review

Das Sprint Review findet am Ende des Sprints statt und dauert zwischen einer und zwei Stunden. Der Product Owner prüft, ob alle Anforderungen vom Team in der Planungssitzung eingegangenen Verpflichtungen richtig umgesetzt wurden. An ihr können auch andere Interessenvertreter wie Endanwender teilnehmen, so dass das Team ein Feedback erhält und auf Verbesserungsvorschläge eingehen kann.[1]

Fehlerhafte oder partiell fertiggestellte Anforderungen sind vom Product Owner abzulehnen und sollten im nächsten Sprint aufgegriffen werden.

[Pichler] empfiehlt folgendes Vorgehen um Fehler zu vermeiden:

Der Product Owner sollte eine aktive Rolle im Sprint Review einnehmen. Er sollte die Arbeitsergebnisse genau betrachten und sich diese gegebenenfalls detailliert erklären lassen. Schließlich ist er maßgeblich am Projekterfolg beteiligt.

Der Product Owner sollte einzelne Teammitglieder nicht herausstellen, da dies das Teamgefühl, die Autonomie und die Selbstorganisation beeinträchtigt.

Probleme und Fehler sollten direkt angesprochen werden.

Wenn das Management am Sprint Review teilnimmt, sollte das umgesetzte Produktinkrement klar und sachlich vorgestellt werden. Auf animierte PowerPoint-Präsentationen und speziell erstellten Prototypen sollte verzichtet werden.

3.3.5 Sprint-Retrospektive

Die Sprint-Retrospektive findet im Anschluss an das Sprint Review statt.[2] Sie dauert meist zwischen eineinhalb und zweieinhalb Stunden und hat das Ziel, die Zusammenarbeit des Teams und die Anwendung des Scrum-Prozesses zu verbessern. Dadurch sollen die Produktivität, die Leistungsfähigkeit und die Softwarequalität steigen.

[Pichler] konnte durch agile Entwicklungspraktiken, neue Teamnormen und verbesserter Verfügbarkeit des Product Owners eine Verbesserung feststellen.

[1] Vgl. [Pichler] Seite 107
[2] Vgl. [Pichler] Seite 111

[Pichler] empfiehlt die *Check-In* Methode.[1] Dabei soll jeder Teilnehmer in einem Satz beschreiben, wie er sich fühlt. So soll der Gesprächseinstieg erleichtert werden.

Im nächsten Schritt schreiben die Teilnehmer positive und negative Erfahrungen aus dem Sprint auf Karteikarten. Die Anzahl sollte auf drei begrenzt werden. Diese werden nach Themen sortiert. An der Anzahl lässt sich bereits das dringlichste Thema erkennen. Nun wird mit der Ursachenforschung des Problems begonnen. Im Anschluss werden Lösungswege zur Problembeseitigung gesucht.

3.3.6 Sprint-Burndown-Bericht

Der Sprint-Burndown-Bericht zeigt den Fortschritt im Sprint auf.[2]

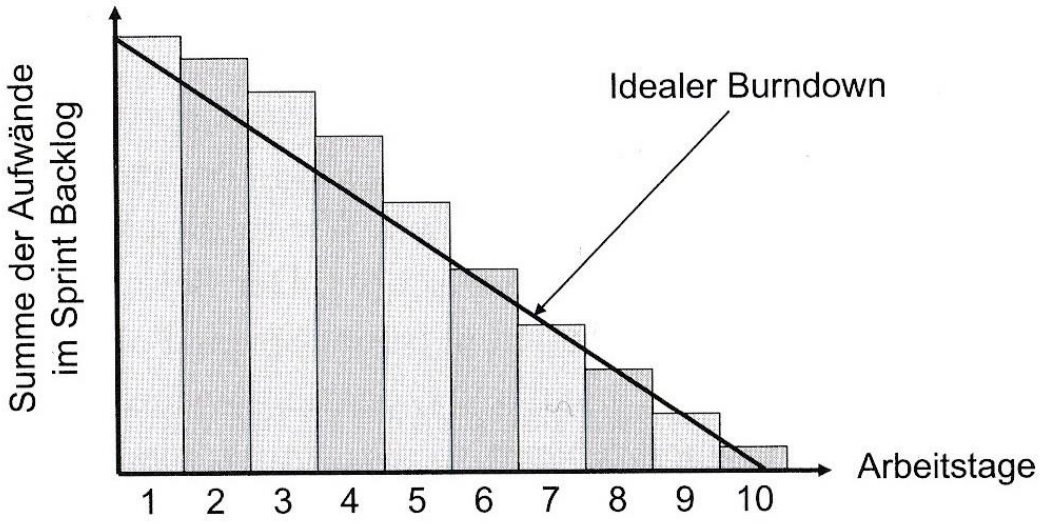

Abbildung 13: Sprint-Burndown-Bericht[Pichler]

Die Abbildung 13 ist wie folgt zu verstehen:

Die y-Achse enthält alle kumulierten Aufwände. Die x-Achse entspricht den Arbeitstagen. Der ideale Burndown beschreibt den idealen Verlauf des Projekts. Man erkennt, dass man sich erst ab dem 6. Arbeitstag im idealen Verlauf des Projekts befindet. Gründe dafür könnten Fehlzeiten von Teammitgliedern oder das Auftreten von Hindernissen sein.

[1] Vgl. [Pichler] Seite 113
[2] Vgl. [Pichler] Seite 117

So lässt sich der Projektfortschritt gut darstellen und jeder kann ersehen, ob der Sprint erfolgreich abgeschlossen wird oder ob Gegenmaßnahmen ergriffen werden müssen, damit man noch im Plan liegt.

3.4 Projekterfolg

Der Chaos-Report der Standish Group[1] ist ein in der Literatur oft zitierter Bericht, der den Erfolg und Misserfolg von IT-Projekten misst.[2] Da der Zugriff auf den Datenbestand kostenpflichtig ist, wurden die Daten aus Wikipedia entnommen.

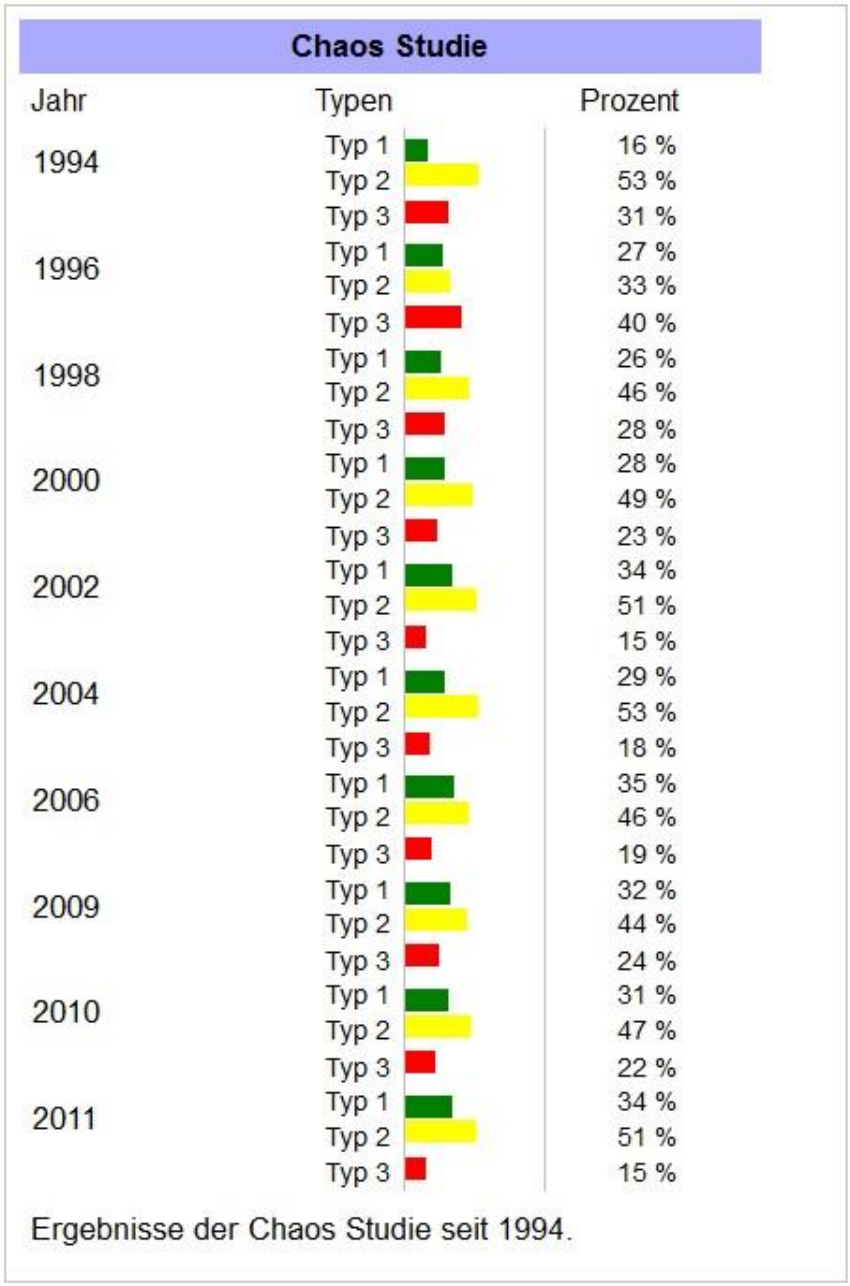

Abbildung 14: Chaos Report aus [WikiChaos]

[1] http://www.standishgroup.com
[2] Vgl. [Hindel] Seite 3

Der Chaos Report unterscheidet drei unterschiedliche Typen:

„Typ 1 - Projekt erfolgreich abgeschlossen: Das Projekt wurde rechtzeitig, ohne Kostenüberschreitung und mit dem ursprünglich geforderten Funktionsumfang abgeschlossen.

Typ 2 - Projekt teilweise erfolgreich: Das Projekt wurde abgeschlossen, es kam jedoch zu Kosten- und/oder Zeitüberschreitungen oder es wurde nicht der vollständige geplante Funktionsumfang erreicht.

Typ 3 - Projekt nicht erfolgreich: Das Projekt wurde abgebrochen.

Die festgestellten Haupterfolgsfaktoren sind:

1. Einbindung der Endbenutzer
2. Unterstützung durch das obere Management
3. Klare Anforderungen

Die Hauptpunkte, die zum Scheitern der Projekte führen sind:

1. fehlende Zuarbeit durch Benutzer
2. unvollständige/unklare Anforderungen
3. häufige Anforderungsänderungen“[1]

Trotz der zunehmenden Verbreitung und Anwendung agiler Vorgehensmodelle in den letzten Jahren, ist kein ausschlaggebender positiver Effekt auf den Projekterfolg erkennbar. Auch heute noch sind nur ein Drittel aller Softwareprojekte „in-time“ und „in-budget“.

[1] [WikiChaos]

4. Oracle Application Express (APEX)

Oracle Application Express (APEX) ist ein deklaratives und webbasiertes Framework zur Entwicklung von Datenbankanwendungen, das innerhalb der Oracle Datenbank läuft.[1]

Es kann in allen Oracle Datenbank-Editionen (EE, SE, XE) installiert werden.[2]

Damit man eine APEX Anwendung entwickeln kann, benötigt man nur einen Web Browser. Über einen Web Browser können APEX Anwendungen administriert, entwickelt und ausgeführt werden.

Ebenfalls bietet Oracle auf der eigenen Internetseite http://apex.oracle.com einen kostenlosen Workspace[3] an. Dazu muss man sich lediglich auf eben erwähnter Internetseite registrieren und nach kurzer Zeit wird der Workspace freigeschaltet. Die dort erstellten Anwendungen dienen lediglich Demonstrationszwecken. Oracle untersagt das Speichern produktiver und sensibler Daten in seinen Richtlinien. Wird der Workspace einige Zeit lang nicht benutzt, so erhält man eine E-Mail mit dem Hinweis, dass der Workspace und alle darin befindlichen Daten bei weiterer Nichtbenutzung gelöscht werden. Es stellt aber eine gute Alternative dar, um sich mit APEX vertraut zu machen, da man keine lokale Oracle Datenbank installieren muss.

[1] Vgl. [Scott] Seite 515
[2] Vgl. [Aust] Seite 145
[3] Arbeitsumgebung

4.1 Geschichte und Entwicklung bis 4.2

APEX hat seinen Ursprung in Oracle WebDB aus dem Jahre 1996.[1] Aus Oracle WebDB wurde Oracle Portal. Der Schwerpunkt änderte sich dabei von datenbankzentrierten Webapplikationen hin zu einer Portal-Lösung. Im Laufe der Zeit änderte sich der Name von Oracle Platform über Project Marvel bis hin zu Oracle HTML DB. Im Jahr 2006 wurde aus Oracle HTML DB schließlich Oracle Application Express (APEX).

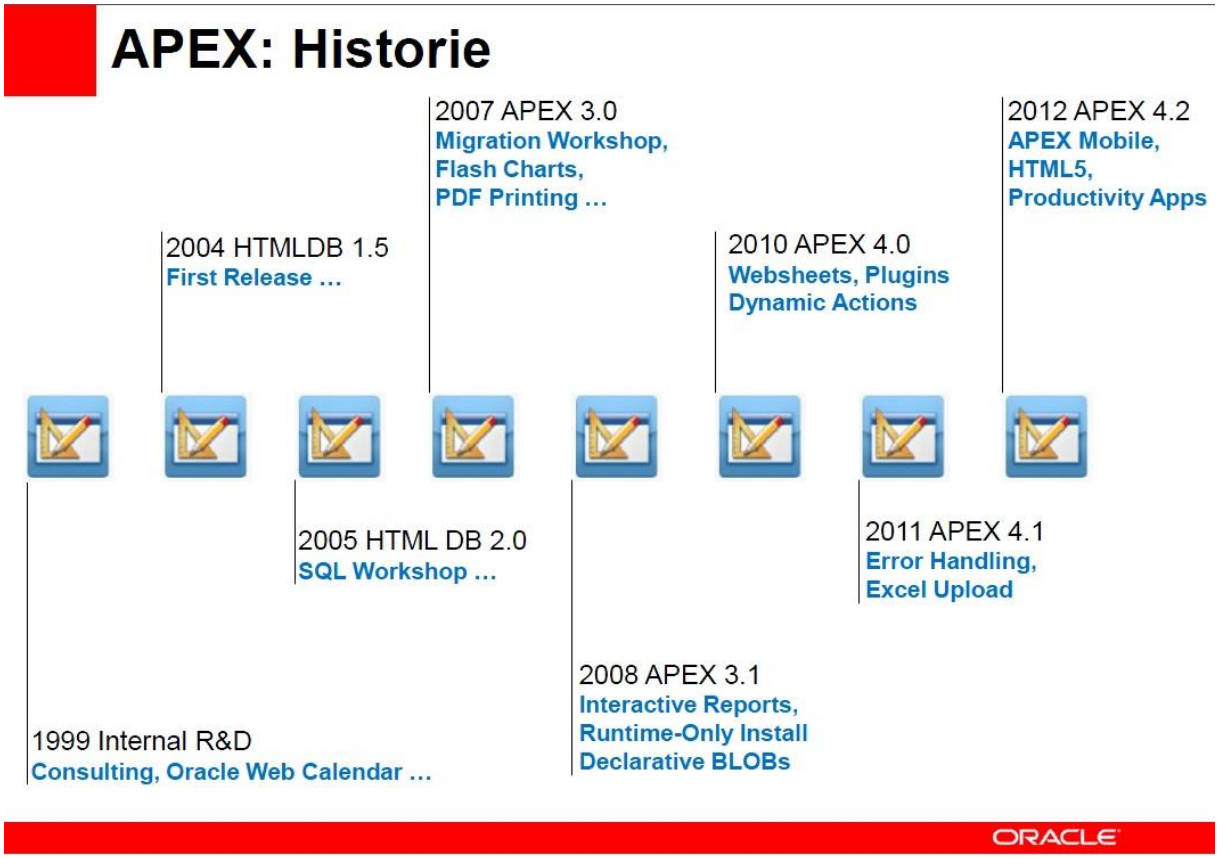

Abbildung 15: APEX Historie [Czarski]

[1] Vgl. [Aust] Seite 145

4.2 Architektur und Technik

APEX läuft innerhalb der Oracle Datenbank und kommuniziert über einen Web Listener mit dem Web Browser.

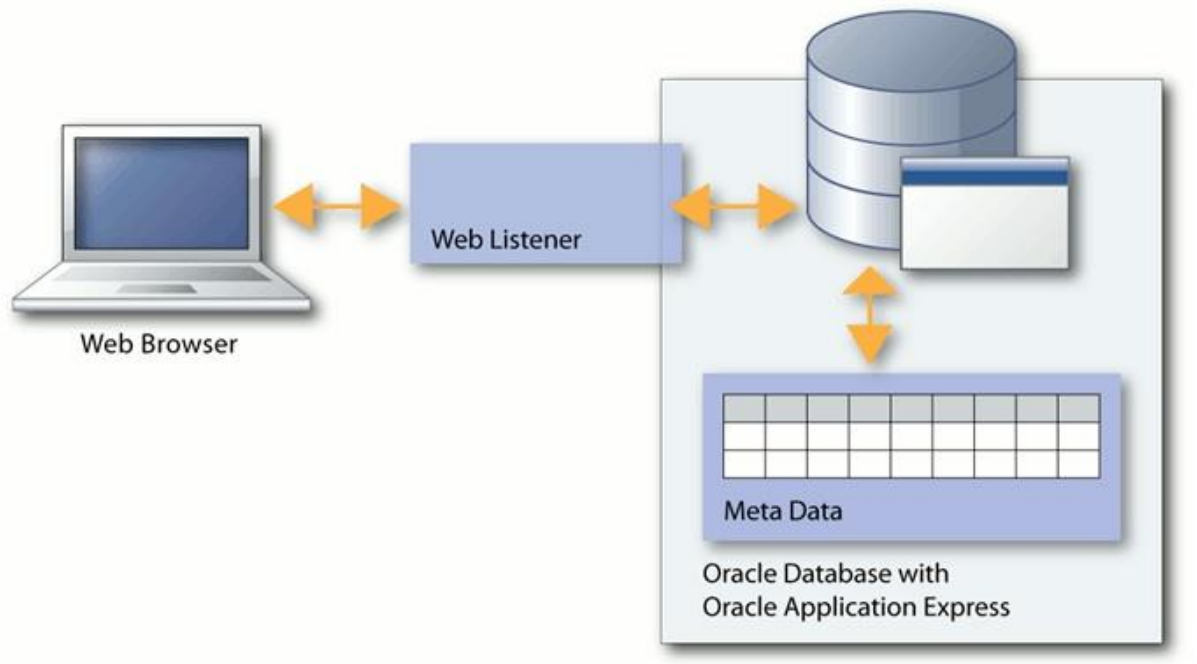

Abbildung 16: APEX Architektur [Oracle2]

Die Applikationsentwicklung findet in APEX durch Senden von HTTP oder HTTPS Anfragen im Web Browser statt.[1] Die Applikationsdefinitionen, also der Aufbau der Applikation mit ihren Funktionalitäten, werden als Meta-Daten in der Datenbank gespeichert. Die Ausführung der Applikation geschieht durch das Auslesen dieser Meta-Daten innerhalb der Datenbank in Echtzeit. Dadurch lässt sich hoher Netzverkehr und Kontextwechsel vermeiden.

APEX besteht aus ungefähr 425 Tabellen und 230 PL/SQL Paketen, die aus mehr als 425.000 Codezeilen bestehen.[2]

[1] Vgl. [Scott] Seite 515
[2] Vgl. [Oracle2]

4.2.1 Kommunikation Web Browser und APEX

Die Kommunikation zwischen Web Browser und APEX kann auf drei unterschiedlichen Arten stattfinden:

Die erste Variante ist der Oracle HTTP Server (OHS). OHS ist ein Web Server, der auf einen modifizierten und optimierten Apache HTTP Server beruht und mittels eines Plung-ins, genannt mod_plsql, den direkten Zugriff auf die Datenbank ermöglicht.[1] Mod_plsql dient zur Herstellung der Datenbankverbindung und dem Aufruf des APEX-Frameworks.[2]

Die URL für den Zugriff auf APEX ist wie folgt aufgebaut:

http://servername:port/pls/apex/f?p=1000:1[3]

Eine mögliche URL könnte so aussehen:

http://fh-koeln.de:7777/pls/apex/f?p=1000:1

Dies ist für die meisten Nutzer etwas kryptisch und lässt sich schlecht einprägen.

In OHS und APEX lassen sich virtuelle Hosts erstellen, so dass man eine freundlichere URL weitergeben kann.[4] Zum Beispiel: http://fh-koeln.de/benutzerverwaltung.

Zudem lassen sich noch einige Performanceverbesserungen in OHS einstellen, um die Serverlast der Datenbank zu minimieren und so für eine bessere Skalierung der APEX Umgebung zu sorgen.

Die erste Möglichkeit ist MPM. MPM steht für Multi-Processing Module und ist eine multi-thread[5] Implementierung, mit der die Datenbankverbindung in einem Pool innerhalb aller Threads[6] geteilt wird.[7]

Die zweite Möglichkeit ist die Web Server Komprimierung. Dabei Komprimiert der Web Server die Inhalte bevor sie zurück gesendet werden, so dass weniger Bandbreite im Netzwerk benötigt wird und dadurch Anfragen anderer Benutzer schneller verarbeitet werden können.[8]

---

[1] Vlg. [Scott] Seite 3
[2] Vgl. [Aust] Seite 153
[3] 1000 gibt die Nummer der APEX Applikation an, 1 steht für die erste Seite dieser APEX Applikation.
[4] Vgl. [Scott] Seite 16
[5] Mehrere Prozesse gleichzeitig.
[6] Prozesse
[7] Vgl. [Scott] Seite 20
[8] Vgl. [Scott] Seite 22

Die dritte Möglichkeit ist die Verwendung von Expiry Headers. Dabei werden im Browser statische Inhalte lokal zwischengespeichert und nicht mehr bei jeder Anfrage neu gesendet.[1]

Die zweite Variante zur Verbindung mit dem APEX-Framework ist das Embedded PL/SQL Gateway (EPG). Im Gegensatz zu OHS läuft EPG vollständig in der Datenbank.[2]

[Aust] empfiehlt aus Gründen der Sicherheit, Performance und des besseren Managements, dass man einen Standart-Apache-HTTP-Server als Proxy zum EPG einsetzt.[3]

Die dritte Variante ist der APEX Listener.[4] Der APEX Listener ist eine J2EE[5] Alternative zum OHS und mod_plsql. Die Datenbankverbindung wird über einen eingebetteten JDBC[6] Treiber hergestellt.

4.2.2 APEX-Engine

Die APEX-Engine ist zuständig für:[7]

- Das Management des Sitzungszustands
- Authentifizierungsservice
- Berechtigungsservice
- Steuerung der Seitenwiedergabe
- Überprüfungsverarbeitung
- Darstellungs- und Seitenverarbeitung

Dadurch, dass die APEX-Engine innerhalb der Oracle Datenbank läuft, profitiert APEX von der Performance und Skalierbarkeit der Oracle Datenbank.

[1] Vgl. [Scott] Seite 30
[2] Vgl. [Scott] Seite 35
[3] Vgl. [Aust] Seite 154
[4] Vgl. [Scott] Seite 50
[5] Java 2 Platform Enterprise Edition
[6] Java Database Connectivity
[7] Vgl. [Oracle2]

4.2.3 Workspaces

Die eigentliche Applikationsentwicklung findet innerhalb eines Workspaces statt, der zuvor in APEX von einem Datenbankadministrator angelegt wurde. Der Workspace enthält ein Datenbankschema oder mehrere Datenbankschemata. In ihm werden die Datenbankobjekte angelegt, wie zum Beispiel Tabellen, Indexe, Views, Synonyme, Prozeduren, Funktionen, Trigger und Constraints.

Für bestimmte Bereiche muss der Datenbankadministrator per Grant Befehl dem Entwickler die nötigen Zugriffsrechte vergeben. Dies ist zum Beispiel bei der Erstellung eines Context Objektes oder beim Zugriff auf ein weiteres Datenbankschema nötig.

In APEX können dem Workspace nun verschiedene Benutzer zugeordnet werden, die sich in den Rollen Administratoren, Entwicklern und Endbenutzern unterscheiden. Jeder angelegte Benutzer darf nur auf den für ihn zugeteilten Workspace zugreifen. Damit man auf mehrere Workspaces zugreifen kann, müssen dort entsprechend neue Benutzer angelegt werden. Ein Entwickler kann nicht gleichzeitig mit einem Account auf zwei unterschiedliche Workspaces zugreifen.

4.3 Das APEX Framework

Das APEX Framework unterteilt sich in vier Bereiche: Application Builder, SQL Workshop, Team Development und Administration.

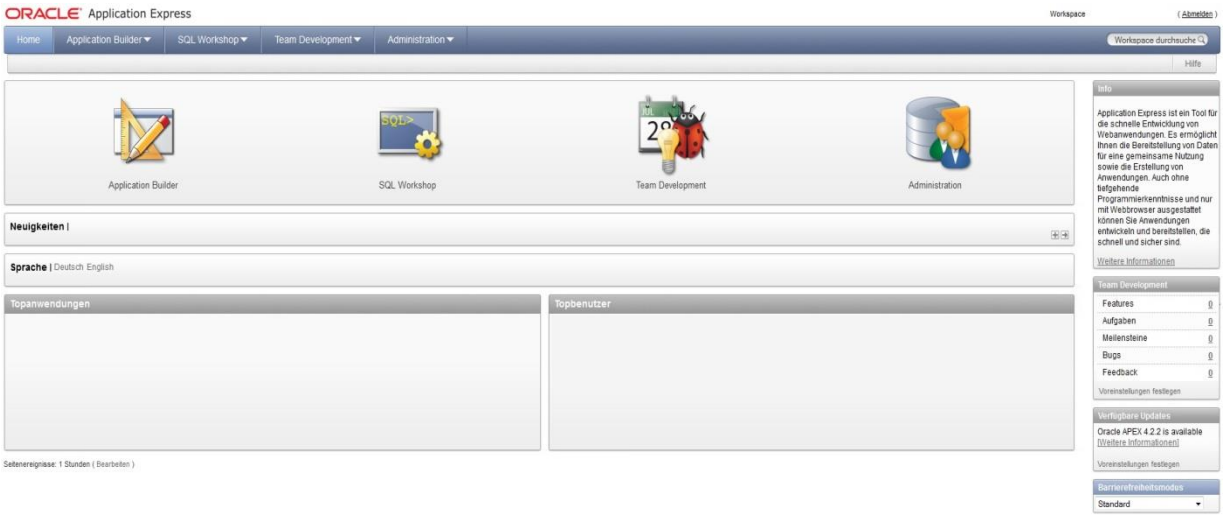

Abbildung 17: APEX Framework

Der Application Builder ist für das Erstellen und Editieren einer APEX Anwendung zuständig. Hier lassen sich erstellte Anwendungen importieren, exportieren oder migrieren. Den Aufbau einer APEX Anwendung beschreibt das Kapitel 4.4.

Im SQL Workshop lassen sich alle Datenbankobjekte anlegen, bearbeiten und entfernen. Hier können auch SQL Befehle und Skripte ausgeführt werden. Über einen Query-Builder lassen sich SQL Abfragen per Point-and-Click auch ohne tiefere SQL Kenntnisse erstellen. Dazu wählt man die entsprechenden Tabellen oder Views aus, verbindet sie über ihre Primär- und Fremdschlüssel und wählt die anzuzeigenden Tabellenspalten aus. Wenn man den Query-Builder nun ausführt, wird die SQL Abfrage automatisch generiert und man sieht sofort das Abfrageergebnis. Darüber hinaus lassen sich auch externe Tools wie der Oracle SQL Developer[1] oder Toad[2] der Firma Quest verwenden.

Der Bereich Team Development unterstützt die Kommunikation und Kollaboration der Entwickler und Endanwender beziehungsweise den Entwicklern und dem Kunden und unterstützt so die Philosophie des Agilen Manifests nach Kommunikation mit dem Kunden

[1] http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html
[2] http://www.quest.com/toad/

und dem Reagieren auf Veränderungen. Hier kann man unter anderem auf neue Features hinweisen, Meilensteine definieren, Bugs und Feedback sammeln oder auf Neuigkeiten aufmerksam machen. Der Endanwender kann auf auftretende Fehler hinweisen und dem Entwicklerteam direkt kommunizieren, ob die geforderten Anforderungen seinen Wünschen entsprechend tatsächlich richtig umgesetzt wurden. Die Anforderungen in Form von user stories lassen sich in Meilensteinen abbilden, so dass man nicht auf externe Tools und deren Verwaltung zurückgreifen muss.

Das APEX Framework unterstützt das teambasierte Entwickeln von Anwendungen.[1] Dazu gibt es zwei Vorgehensweisen. Die erste Möglichkeit ist das Setzen von Seitensperren, so dass kein anderer Entwickler an dieser Seite Änderungen vornehmen darf. Die zweite Möglichkeit ist, dass man auf das *Optimistic Locking* Verfahren von APEX vertraut. Dieser Mechanismus greift, sobald mehrere Entwickler gleichzeitig eine Region einer Seite bearbeiten. Hier kann nur ein Entwickler seine Änderungen abspeichern. Ein anderer Entwickler erhält dann eine Benachrichtigung, dass ein Speichern nicht möglich ist.

Im Bereich Administration kann der Entwickler zum Beispiel ein Schema oder mehr Speicherplatz beim Administrator anfordern. Hier kann der Entwickler auch Benutzer und Gruppen verwalten.

[1] Vgl. [Aust] Seite 194

4.4 Aufbau einer APEX Anwendung

Die Erstellung einer APEX Anwendung ist komplett Wizard gesteuert.

Eine APEX Anwendung besteht aus mehreren HTML Seiten, die über Registerkarten, Schaltflächen oder Links miteinander verbunden sind. Ein Referenzieren der Seiten geschieht durch die entsprechende Seitennummer.

Eine Seite unterteilt sich wiederum in verschiedene Regionen, die von unterschiedlichem Typ sein können: HTML Text, Berichte, Formulare, Diagramme usw.

Innerhalb dieser Regionen lassen sich Seiten Elemente (Page Items) einbinden. Seiten Elemente können sein: Check box, Datumsauswahl, Dateibrowser, Select Listen, List of Values (LOV), Textelemente usw., die die Daten der Datenbank referenzieren.

Schaltflächen in Form von HTML-Buttons dienen der Verzweigung oder dem Ausführen von Prozessen. Ein Prozess führt entweder DML-Anweisungen wie Insert, Update oder Delete oder PL/SQL Code in Form von Prozeduren oder Funktionen aus.

Eine Besonderheit stellt die Seite 0 dar. Alle dort hinterlegten Regionen und Elemente werden auf den folgenden Seiten ebenfalls dargestellt.[1] Hier können gemeinsam genutzte Objekte definiert werden, wie zum Beispiel Navigationspfade, ein zentrales Menü oder ein News-Bereich.[2] [Aust] empfiehlt auf dieser Seite keine teuren SQL-Statements zu verwenden. Mit teuren SQL-Statements ist gemeint, dass man SQL-Abfragen, die einen Full-Table-Scan erzeugen, vermeiden sollte. Dies lässt sich mit dem Cost-Based Optimizer (CBO) feststellen.

Die Geschäftslogik einer APEX Anwendung sollte in der Datenbank in PL/SQL Packeten und nicht in der APEX Anwendung hinterlegt werden.[3] Dies hat zwei Vorteile. Zum einen ist der Code zentral wart bar und muss nicht ständig in den einzelnen Seiten gesucht werden. Dies mag bei kleinen Anwendungen zunächst vernachlässigbar sein, ist aber Vorteilhaft, wenn die Anwendung mit der Zeit wächst. Der zweite Vorteil liegt darin, dass der Code nur einmal in der Datenbank kompiliert wird. Würde der Code innerhalb APEX hinterlegt werden, so würde er nach dem Senden der Seite an die APEX Engine zuerst auf Richtigkeit überprüft und dann neu kompiliert werden, was die Datenbank unnötig belasten würde. Bei Zugriffen von vielen Anwendern macht sich mit einer schlechten Performance bemerkbar. Es ist auch

[1] Vgl. [Aust] Seite 161
[2] Vgl. [Aust] Seite 191
[3] Vgl. [Trivadis] und [Aust] Seite 190

empfehlenswert sich zuvor auf eine einheitliche Namenskonvention zu einigen, damit der Überblick im Datenbankschema gewährt bleibt.

Das Erscheinungsbild einer Anwendung beziehungsweise die Corporate Identity wird durch Templates festgelegt. APEX liefert bereits einige Standard-Templates mit, aus denen man auswählen kann. Eine Anpassung ist über ein Hard Coding oder über CSS möglich. Dazu lassen sich Templates exportieren, editieren und wieder importieren. So kann man unabhängig von der fertigen Anwendung das Erscheinungsbild seiner Anwendung bequem anpassen.

Nach dem Erstellen der APEX Anwendung sollte man diese in verschiedenen Browsern in ihren unterschiedlichen Versionen testen, da Browser sich unterschiedlich verhalten können.

Mobile Anwendungen sind seit einigen Jahren ein immer mehr zunehmender Trend. Seit APEX 4.2 lassen sich über jQuery Mobile[1] sehr einfach mobile APEX Anwendungen erstellen.[2] Dazu muss man bei der Erstellung der APEX Anwendung die Benutzeroberfläche von Desktop auf jQuery Mobile-Smartphone umstellen und kann diese dann mit mobilen Themes personalisieren. Ebenfalls hat man auch Zugriff auf typische Smartphone Funktionen wie das GPS und die Kamera.

[1] http://jquerymobile.com
[2] Vgl. [Mobile]

4.5 JavaScript, jQuery, Ajax, Dynamic Actions und die APEX API

Wie im vorherigen Kapitel bereits beschrieben, besteht eine APEX Anwendung aus unterschiedlichen HTML Seiten. Web Anwendungen mit reinen HTML Seiten haben den Nachteil, dass sie eher statisch sind und somit wenig zur User Experience beitragen.

Damit Webseiten dynamischer dargestellt werden können, unterstützt APEX JavaScript. Der Vorteil von JavaScript ist die client-seitige Verarbeitung von Webinhalten, das heißt, der JavaScript-Code wird direkt im Browser ausgeführt. So können Berechnungen vor dem Senden an die APEX Engine vorgenommen werden, was zu einer Entlastung der Datenbank führt, da nicht ständig bei Aktualisierungen der Seiten diese an die Datenbank gesendet und dort verarbeitet werden müssen.

JavaScript kann in einem Repository der APEX Anwendung vorgehalten werden, so dass jeder Entwickler auf die hinterlegten Skripte zugreifen und diese in seinen Seiten ausführen kann. Der JavaScript-Code wird direkt im Header einer Seite eingefügt und ist sofort beim Seitenaufruf aktiv.

Ein Anwendungsbeispiel ist die automatische Berechnung der Summe einer Bestellung anhand von ausgewählten Produkten.

jQuery[1] ist eine JavaScript-Bibliothek, die Funktionen zur DOM-Manipulation bietet.[2]

Ajax steht für *Asynchronous JavaScript and XML* und stellt ebenfalls eine client-seitige Verarbeitung wie JavaScript dar.[3]

Alternativ zu JavaScript, jQuery und Ajax wurden in APEX 4.0 Dynamic Actions eingeführt. Dies ist ein deklarativer Weg, dynamische APEX Anwendungen auch ohne Kenntnisse in JavaScript, jQuery und Ajax zu erzeugen.[4]

[Scott] zeigt drei Probleme bei der Verwendung von JavaScript auf:[5]

- Der Code kann an verschiedenen Orten hinterlegt werden: in einer externen Datei, in der HTML Region, im Region Header, Teil eines Seitenprozesses, usw. Dies erschwert das Auffinden des Codes, wenn man ihn debuggen oder ändern möchte.

[1] http://jquery.com/
[2] [WikijQuery]
[3] [WikiAjax]
[4] Vgl. [Oracle3]
[5] Vgl. [Scott] Seite 310

- Bei der Entwicklung innerhalb eines Teams können Entwickler unterschiedliche Techniken verwenden und der Code könnte nicht wiederbenutzbar sein.
- Nicht alle APEX Entwickler verstehen JavaScript.

Zudem nennt [Scott] drei Vorzüge bei der Verwendung von Dynamic Actions:[1]

- Dynamic Actions sind deklarativ. Somit wissen Entwickler genau wo der Code hinterlegt ist, nämlich innerhalb einer Seite.
- Dynamic Actions haben ein eingebautes Framework, das die Pflege der Konsistenz applikationsübergreifend erlaubt.
- Dynamic Actions ermöglichen einfache Event-basierte Aktionen für nicht-JavaScript Entwickler.

Da JavaScript auch oft Sicherheitsrisiken für den Anwender darstellen können und einige Anwender deshalb JavaScript in ihrem Browser deaktivieren, ist die Verwendung von Dynamic Actions vorzuziehen.

Die APEX API enthält nützliche Funktionen für den Entwickler. Im Folgenden wird ein kurzer Überblick über die vorhandenen Pakete gemacht. Sie wurden vom Verfasser im Original aus [Oracle4] übernommen. Detaillierte Informationen lassen sich aus [Oracle4] entnehmen.

- APEX_APPLICATION
  Use the APEX_APPLICATION package to take advantage of many global variables.
- APEX_APPLICATION_INSTALL
  The APEX_APPLICATION_INSTALL package provides many methods to modify application attributes during the Application Express application installation process.
- APEX_AUTHENTICATION
  The APEX_AUTHENTICATION package provides a public API for authentication plugins.
- APEX_COLLECTION
  Use APEX_COLLECTION to temporarily capture one or more nonscalar values. You can use collections to store rows and columns currently in session state so they can be accessed, manipulated, or processed during a user's specific session.
- APEX_CSS
  The APEX_CSS package provides utility functions for adding CSS styles to HTTP output. This package is usually used for plug-in development.

[1] Vgl. [Scott] Seite 311

- APEX_CUSTOM_AUTH
  Use the APEX_CUSTOM_AUTH package to perform various operations related to authentication and session management.
- APEX_DEBUG
  The APEX_DEBUG package provides utility functions for managing the debug message log.
- APEX_ESCAPE
  The APEX_ESCAPE package provides functions for escaping special characters in strings, to ensure that the data is suitable for further processing.
- APEX_ERROR
  The APEX_ERROR package provides the interface declarations and some utility functions for an error handling function and includes procedures and functions to raise errors in an Application Express application.
- APEX_INSTANCE_ADMIN
  The APEX_INSTANCE_ADMIN package provides utilities for managing an Oracle Application Express runtime environment. Use the APEX_INSTANCE_ADMIN package to get and set email settings, wallet settings, report printing settings and to manage scheme to workspace mappings.
- APEX_IR
  The APEX_IR package provides utilities you can use when programming in the Oracle Application Express environment related to interactive reports.
- APEX_ITEM
  Use the APEX_ITEM package to create form elements dynamically based on a SQL query instead of creating individual items page by page.
- APEX_JAVASCRIPT
  The APEX_JAVASCRIPT package provides utility functions for adding dynamic JavaScript code to HTTP output. This package is usually used for plug-in development.
- APEX_LANG
  Use APEX_LANG API to translate messages.
- APEX_LDAP
  Use APEX_LDAP to perform various operations related to Lightweight Directory Access Protocol (LDAP) authentication.
- APEX_MAIL
  Use the APEX_MAIL package to send an email from an Oracle Application Express application.
- APEX_PLSQL_JOB

Use APEX_PLSQL_JOB package to run PL/SQL code in the background of your application. This is an effective approach for managing long running operations that do not need to complete for a user to continue working with your application.

- APEX_PLUGIN
  The APEX_PLUGIN package provides the interface declarations and some utility functions to work with plug-ins.
- APEX_PLUGIN_UTIL
  The APEX_PLUGIN_UTIL package provides utility functions that solve common problems when writing a plug-in.
- APEX_UI_DEFAULT_UPDATE
  You can use the APEX_UI_DEFAULT_UPDATE package to set the user interface defaults associated with a table within a schema. The package must be called from within the schema that owns the table you are updating.
- APEX_UTIL
  Use the APEX_UTIL package to get and set session state, get files, check authorizations for users, reset different states for users, and also to get and set preferences for users.
- APEX_WEB_SERVICE
  The APEX_WEB_SERVICE API enables you to integrate other systems with Application Express by allowing you to interact with Web services anywhere you can use PL/SQL in your application. The API contains procedures and functions to call both SOAP and RESTful style Web services.
- JavaScript APIs
  Use these JavaScript functions and objects to provide client-side functionality, such as showing and hiding page elements, or making XML HTTP Asynchronous JavaScript and XML (AJAX) requests.

5. APEX Alternativen

Web-Anwendungen mit Datenbank zentrischer Ausrichtung lassen sich auch mit HTML-Editoren und JavaScript realisieren. Dies würde aber einen erheblichen Mehraufwand in der Entwicklung bedeuten, da man die in APEX bereits vorhandenen Funktionen zuerst einmal entwickeln und implementieren muss.

Eine Alternative zu APEX ist das Framework Formspider.[1] Im Gegensatz zu APEX benötigt ein Entwickler kein Wissen über JavaScript, jQuery oder Ajax, da die Erstellung einer Anwendung nur über PL/SQL erfolgt. Das Formspider Framework ist ebenfalls rein Web-basiert. Zudem ist noch ein Datenbank Tool wie der SQL-Developer oder Toad für die Pflege von SQL Packages notwendig.

Gegenüber APEX 4.2 konnten in der aktuellen Formspider Version 1.4.0 folgende Nachteile identifiziert werden:

- Keine Unterstützung von Team Development
- Keine Unterstützung von Versionskontrollen
- Kein GUI Designer
- Keine Formatvorlagen Unterstützung
- Lizenzgebühr

Die Vorteile gegenüber APEX 4.2 sind:

- Nur PL/SQL Kenntnisse erforderlich
- Mehrstufige Maste-Detail Relationen
- Transaktions Management

[1] Vgl. [Formspider]

6. Fazit

Generell bedürfen Web-Anwendungen keiner langen Planungsphase wie in klassischen Vorgehensmodellen üblich, da sie vor allem in APEX sehr schnell realisiert werden können und für den Endanwender einsatzbereit sind.

APEX eignet sich hervorragend beim Einsetzen eines agilen Vorgehensmodells wie Scrum. Alle Punkte des agilen Manifests lassen sich problemlos umsetzen. Ein Prototyping ist in APEX nicht notwendig, so dass man am Ende eines Sprints immer funktionsfähige und für den Endanwender produktiv nutzbare Software erhält. Durch das Feedback-System in APEX können Entwickler sofort auf Kundenänderungen reagieren. Ebenfalls wird so die Zusammenarbeit mit dem Kunden gestärkt.

Ein Einsatz des Extreme Programming ist mit Einschränkungen denkbar, da die testgeriebene Entwicklung trotz des vorhanden Tools DB-Unit unter Datenbankentwicklern nicht sehr weit verbreitet ist. Extreme Programming spielt seine Vorteile vor allem bei objektorientierter Programmierung aus.

Jedoch ist eine Mischung einzelner Aspekte, wie zum Beispiel das Pair-Programming aus Extreme Programming innerhalb des Scrum Prozesses denkbar.

Kanban kann ebenfalls zum Scrum Prozess hinzugezogen werden und so den Projekterfolg für das Entwicklerteam visualisieren. Zu dem schützt es das Entwicklerteam durch die Limitierung des Work-In-Progress vor Überlastungen.

Abschließend lässt sich festhalten, dass sich agile Prozesse sehr gut im APEX Umfeld eignen. In welcher Form oder Mischform sollte immer projektabhängig untersucht werden.

7. Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
APEX	Application Express
API	Application Programming Interface
CASE	Computer-Aided Software Engineering
CBO	Cost-Based Optimizer
CSS	Cascading Style Sheets
DML	Data Manipulation Language
DOM	Document Object Model
EE	Enterprise Edition
EPG	Embedded PL/SQL Gateway
GPS	Global Positioning System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
J2EE	Java 2 Platform Enterprise Edition
JDBC	Java Database Connectivity
LDAP	Lightweight Directory Access Protocol
LOV	List of Values
MPM	Multi-Processing Module
OHS	Oracle HTTP Server
PL/SQL	Procedural Language/Structured Query Language
ROI	Return on Investment
RUP	Rational Unified Process
SE	Standard Edition
SOAP	Simple Object Access Protocol
TDD	Test-Driven Development
UML	Unified Modeling Language

URL	Uniform Resource Locator
WIP	Work in Progress
XE	Express Edition
XML	Extensible Markup Language
XP	Extreme Programming

8. Literaturverzeichnis

[Aust] Aust, D. et al.: Oracle APEX und Oracle XE in der Praxis, 1. Aufl., u.a. Heidelberg 2010.

[Czarski] Czarski, C.: Application Express: Stand der Dinge und Ausblick auf Version 5.0, http://www.doag.org/de/events/konferenzen/doag-2013-development/das-programm/vortragsunterlagen-zum-download.html, aufgerufen am: 01.07.2013.

[Formspider] o.V., o.J.: Formspider, http://theformspider.com/, aufgerufen am: 01.08.2013

[Glogler] Gloger, B.: Scrum – Produkte zuverlässig und schnell entwickeln, 3. Aufl., München 2011.

[Hindel] Hindel, B. et al.: Basiswissen Software-Projektmanagement, 1. Aufl., Heidelberg 2004.

[IT-Agile1] o.V., o.J.,it-agile: Was ist Crystal?, http://www.it-agile.de/crystal.html, aufgerufen am: 01.07.2013.

[IT-Agile2] o.V., o.J.,it-agile: Kanban, http://www.it-agile.de/kanban.html, aufgerufen am: 01.07.2013.

[IT-Agile3] o.V., o.J.,it-agile: Testgetriebene Entwicklung (TDD), http://www.it-agile.de/wasisttdd.html, aufgerufen am: 20.07.2013.

[IT-Agile4] o.V., o.J.: it-agile: Was ist FDD?, http://www.it-agile.de/fdd.html, aufgerufen am: 30.07.2013.

[Ludewig] Ludewig, J. und Lichter, H.: Software Engineering – Grundlagen, Menschen, Prozesse, Techniken, 1. Aufl., Heidelberg 2007.

[Mobile] o.V., o.J.: APEX-Anwendungen für mobile Endgeräte, http://apex.oracle.com/pls/apex/ws?p=49675, aufgerufen am: 26.07.2013.

[Oracle1] o.V., o.J., Oracle Application Express, http://www.oracle.com/technetwork/developer-tools/apex/overview/index.html, aufgerufen am: 20.07.2013.

[Oracle2] o.V., o.J., Oracle Application Express Architecture, http://www.oracle.com/technetwork/developer-tools/apex/apex-arch-086399.html, aufgerufen am: 20.07.2013.

[Oracle3] o.V., o.J.: AJAX-Technologie ganz einfach: Dynamic Actions, http://www.oracle.com/webfolder/technetwork/de/community/apex/tipps/apex40-dynamicactions/index.html, aufgerufen am: 30.07.2013.

[Oracle4] o.V., o.J.: Oracle Application Express API Reference Release 4.2, http://docs.oracle.com/cd/E37097_01/doc/doc.42/e35127/toc.htm, aufgerufen am: 30.07.2013.

[Pichler] Pichler, R.: Scrum – Agiles Projektmanagement erfolgreich einsetzen, 1. Aufl., Heidelberg 2008.

[Scott] Scott, J.E. et al.: Expert Oracle Application Express, 2011.

[Sommerville] Sommerville, I.: Software Engineering, 6. Aufl., München 2001.

[Trivadis] o.V., 2011: Oracle Application Express Tipps für Entwicklung und Betrieb, http:// http://www.trivadis.com/uploads/tx_cabagdownloadarea/APEX_Coding_Guidelines_1_0_HiRes.pdf, aufgerufen am: 01.07.2013.

[WikiAjax] o.V., o.J.: Ajax (Programmierung), http://de.wikipedia.org/wiki/Ajax_(Programmierung), aufgerufen am: 30.07.13.

[WikiChaos] o.V., o.J.: Chaos-Studie, http://de.wikipedia.org/wiki/Chaos-Studie, aufgerufen am: 30.07.2013.

[WikiFDD] o.V., o.J.: Feature Driven Development, http://de.wikipedia.org/wiki/Feature_Driven_Development, aufgerufen am: 30.07.2013.

[WikijQuery] o.V., o.J.: jQuery, https://de.wikipedia.org/wiki/JQuery, aufgerufen am: 30.07.2013.

[WikiRup] o.V., o.J.: Rational Unified Process, http://de.wikipedia.org/wiki/Rational_Unified_Process, aufgerufen am: 30.07.2013.

[WikiXP] o.V., o.J.: Extreme Programming, http://de.wikipedia.org/wiki/Extreme_Programming, aufgerufen am 30.07.2013.

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt beziehungsweise in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Bergheim, 8. August 2013

Artur Kusmierczyk